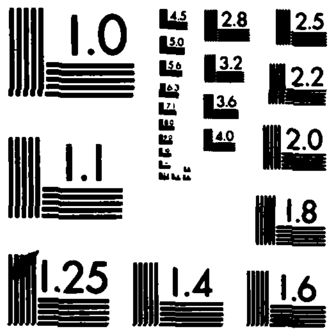
AD-A138 160 AN AUTOMATED COMPUTER COMMUNICATION NETWORK PROTOCOL VERIFICATION SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING 1/3

UNCLASSIFIED K R MARTIN DEC 83 AFIT/GCS/EE/83D-12 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

DTIC

1

An Automated Computer Communication Network Protocol Verification System

Thesis

AFIT/GCS/EE/83D Kenneth R. Martin Jr.

Capt. USAF

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/83D-12

An Automated Computer Communication Network Protocol Verification System

Thesis

AFIT/GCS/EE/83D Kenneth R. Martin Jr.

Capt. USAF

AFIT/GCS/EE/83D-12

AN AUTOMATED COMPUTER COMMUNICATION PROTOCOL VERIFICATION SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air Training Command
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Kenneth R. Martin Jr.
Capt. USAF
Graduate Computer Science

December 1983

Preface

A lot of emphasis has been placed on developing methods to validate computer communication network protocols. One method develped concerns modeling of protocols with graphical techniques. This report describes the development and implementation of an automated protocol evaluation tool using the Program Process Modeling Language as the means of specifying the protocol and the Evaluation Net(E-Net) as the underlying graphical model. The Program Process Modeling Language(PPML) was developed by Dr. W. E. Riddle to specify a protocol for analytical analysis and the E-Net was developed byl Dr. Gary Nutt as a means of modeling computer systems for performance analysis.

Thanks are due to Maj. Walter Seward who sponsored this investigation. His patience and guidane throuthout this project proved invaluable. Thanks are also in order to Dr. Gary Lamont for his participation as a reader of this report. Special thanks are given to friends and classmates for their encouragement and support throughout this endeavor.

Contents

List of Figures

Abstract

An automated tool for computer network communication protocol validation was developed and implemented. The method utilyzes the Program Process Modeling Language (PPML) to specify the protocol and an automated procedure to convert the PPML description into an equivalent Evaluation Net in order to evaluate the protocol. Simulation techniques are used to exercise the Evaluation Net presenting data on message transmission times and global state generation.

List of Abbreviations

PPML	Program Process Modeling Language
E-Net	Evaluation Net
HDLC	High-level Data Link Control

AN AUTOMATED COMPUTER COMMUNICATION NETWORK PROTOCOL VERIFICATION SYSTEM

I INTRODUCTION

Computer networks rely on the proper operation of both hardware systems and software systems in order to satisfy their function. Much study has been given to the reliability and function of the hardware associated with computer systems since their inception. Until recently though, little effort has been expended in the study of software reliability and function. This was probably due to the fact that there were not many tools designed to attack this issue. Lately, we have seen a large emphasis placed on the problem of software design and development for the end purposes of increased reliability and functional correctness. These concepts have been incorporated into the new discipline known as Software Engineering.

Now that there are tools for the evaluation of software systems the problem of evaluating computer network protocols has risen to the forefront. It has been realised that the software involved in computer networks has a large impact on the operation of the network. This can be demonstrated by the fact that many of the failures of computer networks have been attributed to the software(Ref 14:3).

Until recently the analysis of computer network protocols has been accomplished primarilly by manual development and analysis of models of the protocols. The models, whether graphical or algorithmic were built by hand and subsequently analyzed manually. This means of evaluating protocol performance was adequate as long as the protocols were not very complex. The point has been reached where purely manual analysis is not feasible. Increasing emphasis is being placed of the development of automated methods for the design and analysis of computer network protocols(Ref 14). This area of research is the subject of this paper. The objective of this project is the design and analysis of a system for the automatic validation of computer network protocols.

PROBLEM STATEMENT

The United States Air Force and the computer networking community at large have demonstrated the need for an automated system for the specification and verification of computer communication protocols. The complexity of computer networks has steadily increased over the past ten years. The increase in complexity has resulted in more complicated procedures for computers to communicate with each other. The point has been reached where it is impractical to rely totally on human protocol specification and verification. The computer has been suggested as one tool to help relieve the reliance on human performance in these areas.

SCOPE

The intent of this project is to develop a computer assisted means of validating computer communication protocols. The model will be based on the E-Net as developed by Gary Nutt(Ref 8). The reason for choosing the E-Net was two fold. Firstly, the E-Net has the capability to model systems that operate concurrently and asynchronously. Secondly, the E-Net has provisions for evaluating the performance aspects of a system in terms of time. This technique will be applied to computer communication protocols to verify their correctness and to detect unwanted features. The primary detrimental characteristic to be detected is the deadlock. There are other, more subtle problems that could occur in a computer network and the incorporation of procedures to detect them will be left for further investigation.

ASSUMPTIONS

The primary assumption during this effort will be that the communicating processes communicate via the transmission of messages over a media that is imperfect. This means that the media will degrade the performance of the network by introducing errors in the messages and in some cases will competely lose the message. This assumption adds to the complexity of the model and its analysis. Since these characteristics of transmission media are probablistic in nature a probablistic model of them will have to be developed.

APPROACH

The development of this project will proceed in a top down manner as far as possible. Currently accepted techniques of Software Engineering will be used to define and design the system(Ref 9). First the specification of the system will be developed. This will include a description of how the user will interface with the system and what services will be expected to be provided. The testing procedure will also be developed during this phase.

The next phase will entail the design of the system. This will proceed from the highest level of user interface to the lowest level of module development. The primary means of achieving the design will be through the software engineering techniques resulting in a top down analysis and design.

ORGANIZATION

Chapter II presents the results of an analysis of currrently available protocol validation methods. Chapter III presents the requirements definition for an automated protocol validation tool. Chapter IV contains a description of the overall design of the system as well as description of the detailed design. Chapter V describes the implementation aspects of the project. Chapter VI presents the testing procedure and its results and Chapter VII will summarize the results of the project and presents recommendations for further work.

II Analysis

Introduction

The first steps of the software development cycle are concerned with data collection and analysis(Ref 8:13). these steps manifested themselves in the form of a search of the current literature on the subject of computer communication network protocol validation. The need for more formal methods in this area has led to researchers directing their attention to the solutiou of the problem. Many articles have resulted. The scope of the research included the IEEE proceedings and other journals on computer communication networks available through national informational institutions.

The remainder of the chapter will discuss protocols and techniques that have been developed to evaluate them. A detailed discussion will be given covering the method that was chosen for this project, the Program Processing Modeling Language and the E-Net.

Protocol Characteristics

When a protocol validation effort is begun there are certain characteristics that the validation effort is aimed at detecting. A description of the primary ones follows.

1) Deadlock freeness: No abnormal terminal state.
2) Liveness: From each reachable state any other reachable state is reachable.
3) Tempo-blocking: No non productive infinite looping.

4) Starvation freeness: No process will be prevented forever from acquiring a resource it needs.

5) Failure recovery: After a failure the protocol will return to normal operation within a finite number of steps.

6) Self synchronization: From any abnormal state, the protocol will return to a normal state within a finite number of steps.

7) Correct execution of the purpose of the protocol. (Ref 6:1678)

Protocol Validation Techniques

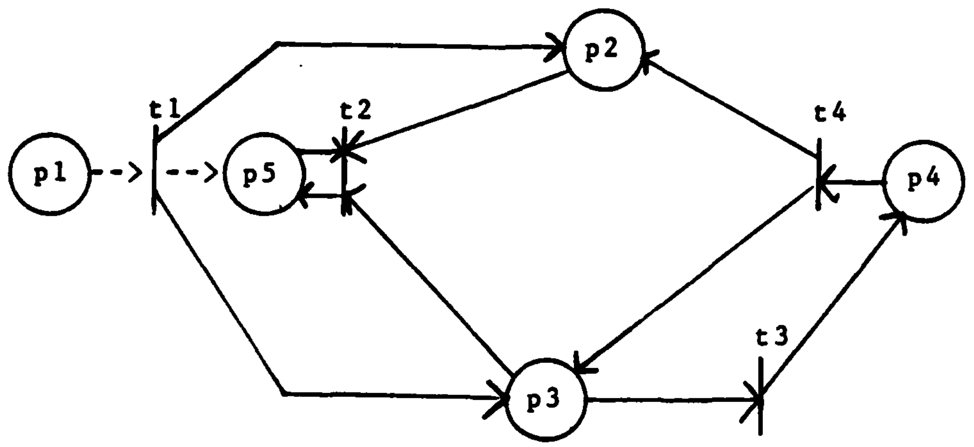
Protocols have been described as the rules governing the exchange of messages between a system of cooperating processes(Ref 6:1671). Protocol validation is the process of examining a protocol to insure that a system satisfies its design specification and operates to the satisfaction of its users(Ref 3:624). Several techniques have been developed to attack the problem of protocol validation. Early validation efforts were primarily accomplished by "walk throughs" and scenario analysis. These manual methods often resulted in specification errors going undetected resulting in system degradation or failure(Ref 14).

The failure of these "ad hoc" validation attempts made it evident that more rigorous and structured methods were needed to adequately accomplish the validation effort. The basic technique that has been applied to this problem is modeling. Several different modeling techniques have been developed to include Finite State Machines(FSM), Petri Nets, and high level programming languages. A summary of each of these techniques follows.

Finite state machines were used quite early in the protocol validation effort(Ref 3:626). The states of the system are represented by nodes and transitions from one state to another are represented by directed arcs between nodes. The action of the system is modeled by generating all possible states of the system. This method is applicable to simple topology systems, such as two party systems and to systems with a small number of states. The advantage of this technique is that global properties of the protocol can be checked. This technique has been used to model and validate several actual protocols(Ref 6:1673).

Another method that uses the global state generation to analyze a systems' performance is the Petri net. Petri nets were originally designed to model systems with interacting concurrent components. Carl A. Petri developed the original theory and delivered his results in his doctoral thesis in 1962(Ref 11:3). Petri nets have a wider applicability as a model of communication systems due to their ability to more easily represent systems that have a possibility of an infinite number of states(Ref 6:1673). Petri nets are made up of several basic entities; places, transitions, directed arcs and tokens. The set of places is related to the transition by two functions, the input function and the output function. The state of the system is represented by the placement of tokens in the places. Tokens travel through the net as a consequence of the firing of the transitions. A Petri net can be graphically represented by places being depicted as circles and transition as bars (Fig 2.1). The places are connected to the transitions by directed arcs. A place connected to a transition by an arc going from the place to the transition is an input place to the transition and a place connected to a transition by an arc going from the transition to the place is an output place of the transition. There may be any number of input places to a transition and any number of output places from a transition. A transition is enabled when each of its input places has at least one token in it. When a transition is enabled it may fire. There is no constraint as to when the transition has to fire after it is enabled. In particular, it does not have to fire immediately after becoming enabled. When a transition fires a token is removed from each of its input places and a token is deposited in each of its output places. The global state generation is represented by recording the token placement after a transition has fired. This record of states is called the "token machine". The basic Petri net can model in a nondeterministic way the action of a system. The nondeterminism arises due to the rules associated with the transition firings. For instance, if a place is shared by two enabled transitions either transition may fire (Fig 2.2). Another aspect of the Petri net that detracts somewhat from its utility is the form of the tokens. The

a) Graphical Representation



b) Formal Structure of the above Petri Net

```
              C = (P,T,I,O)
              P = {p1,p2,p3,p4,p5}
              T = {t1,t2,t3,t4}

I(t1) = {p1}                O(t1) = {p2,p3,p5}
I(t2) = {p2,p3,p5}          O(t2) = {p5}
I(t3) = {p3}                O(t3) = {p4}
I(t4) = {p4}                O(t4) = {p2,p3}
```

I represents the input function and
O represents the output function.

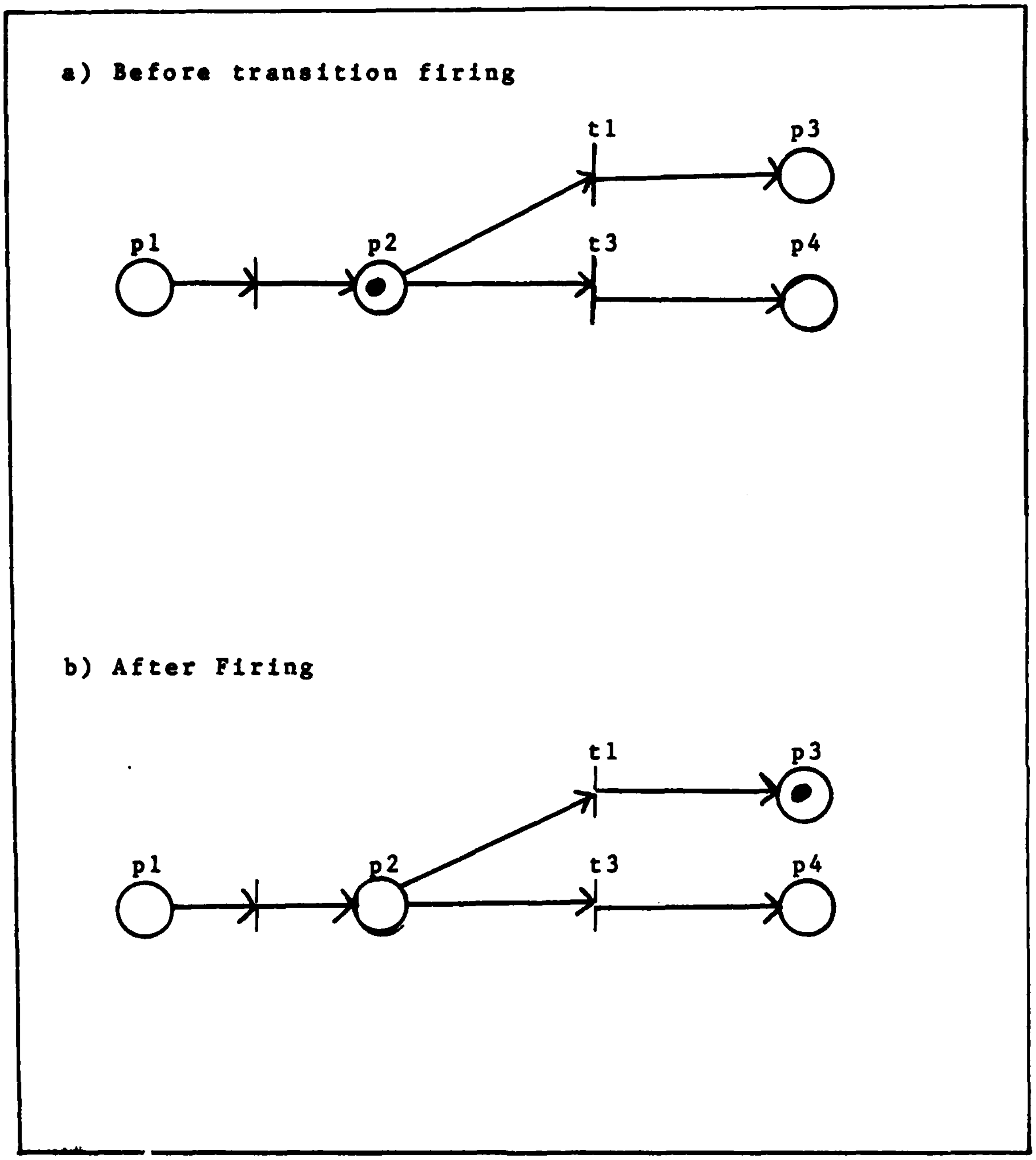
Example of A Petri Net

Figure 2-1

tokens do not have any unique identifying features and are therefore indistinguishable from each other. We would like to be able to differentiate between various types of messages being processed by the system. This corresponds to having the capability to distinguish one token from another.

Extensions to the basic Petri net have been developed by several researchers to enhance the usefulness of Petri nets. The concept of a finite time associated with a firing is incorporated in the Time Petri net(Ref 7:1036). The ability to use distinguishable tokens is introduced by the Numerical Petri net(Ref 15 and 16). The Evaluation net(E-Net) incorporates many of the extended features that have been developed. The ideas of distinguishable tokens, finite firing times and decision making by transition make it a powerful tool for modeling communication systems(Ref 8). A more detailed description of the E-Net will be given in a later section.

The second major class of models used to validate protocols is high level languages. These include programming languages and formal grammars. Formal grammars attempt to define the allowable sentences of a language(Ref 14). Utilyzing the theories of formal languages and finite state automata a protocol is modeled by defining what messages can be sent through a set of production rules that describe the evolution of the communication process(Ref 5).



Transition Firing and Conflict

Figure 2-2

Programming language models use the concept of assertion proving to validate protocols. The protocol is first described in a programming language, like Pascal, and then the protocol is validated using assertion proving theory on the programming language model(Ref 2,13). The programming language model is attractive to protocol designers because once the protocol has been shown not to have undesirable characteristics it can be easily implemented from the programming language model. The main drawback to this approach is that the underlying validation method is the assertion proving technique. This process remains largely accomplished by hand analysis and is not readily adaptable to computer implementation although research is being done to remove this limitation.

Validation Methods

The two basic methods of validation are reachability analysis using global state generation and analytical proofs of assertions or inductive proofs. The graphical models use reachability analysis and the high level language models use the methods of proofs.

The graphical models validate protocols by first generating a reachability tree from the global state machine of the system. The reachability tree is then analysed to detect any undesirable characteristics present in the protocol. One of the primary advantages of this method is that the global state generation and reachability tree analysis can be automated to a great extent(Ref 16:35).

This means of validation has been described as being most applicable to modeling the control aspects of the system, i.e. that certain events will or will not occur(Ref 6:1678).

The method of proofs used by the high level language models involves the techniques of either assertion proving in programming language models or inductive proofs in the formal language models. These methods are well suited to validating the data transfer characteristics of a protocol(Ref 6:1678).

After examining the available protocol validation methods it became evident that the ones most applicable to computer implementation were the graphical models of finite state machines and Petri nets. In order to do a comprehensive evaluation of a protocol some means of evaluating the performance of the protocol should also be included in the validation system. The reason for including this feature is that the time to transmit a message through the network may detract from the usefullness of the network. With this in mind, It was decided that the best means available for the purpose was the E-Net.

At the present time there are no automated methods of generating an E-Net from a description of the protocol. What was needed was a way to specify the protocol so that an E-Net could be generated. After the E-Net was built it would be exercised resulting in the output of a token machine and information on transmission times of messages. The first step was to find a way to specify the protocol.

Program Process Modeling Language

W.E. Riddle has developed a means of describing systems composed of independent asynchronously operating sequential processes(Ref 12). The processes communicate with each other by the transmission of messages. Each process is described by a program consisting of statements derived from a special purpose language. The usefulness of this model comes from the fact that J.L. Peterson has shown that the process descriptions could be converted to equivalent Petri nets(Ref 10). The language developed by Riddle is called the Program Process Modeling Language(PPML). A summary of the basic statements in the PPML follows.

1) SET T: Place a message of type "T" in the message buffer. Where "T" is the name associated with a particular message type.
2) SEND L: Send the message in the message buffer to link process "L". Where "L" is the name of a particular communication link. A communication link corresponds to a one way transmission line.
3) RECEIVE L: Request a message from link process "L". Wait if necessary until one is received.
4) UNLESS T S: Test the type of message in the message buffer and branch to "S" unless the messsage is of type "T".
5) IF-INTERNAL-TEST S: An internal, data dependent test is performed and either continue to the next

instruction or branch to statement "S".

6) GO-TO S: Transfer control to statement "S".

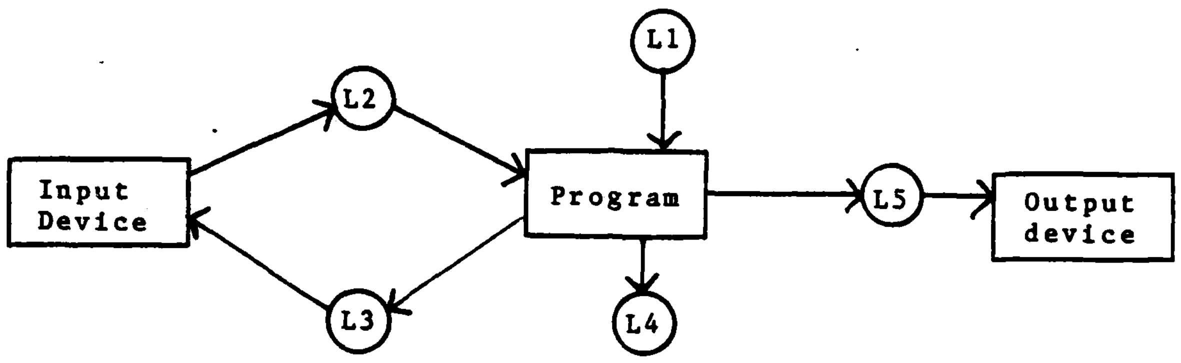
7) END: Terminate the process.(Ref 12).

There are more statements defined in the PPML but these are the only ones necessary to describe a system(Ref 12). The PPML statements not described are used to ease the specification of the communicating processes. They include looping constructs and procedures to specify more than one link process in a single statement. Figure 2-3 describes how a system can be modeled using the PPML.

Peterson has developed a procedure to build a Petri net from the PPML(Ref 11:220). He models only the SEND and RECEIVE statements and represents the sequencing by determining which statements can precede any other statement. This process is well suited to transforming the PPML description into a Petri net and is instructive with respect to its procedures. Unfortunately, The process is not directly applicable generation of E-Nets due to the differences between the Petri net and the E-Net. In order to accomodate these differences, several syntactical as well as semantical changes to the PPML had to be made before it could be incorporated into the final design. The changes will be discussed in the Chapter 4 of this report.

Evaluation Net

The Evaluation Net(E-Net) was developed by Gary Nutt in 1972 as a means of modeling systems composed of asynchronously operating systems for the primary purpose of



Program

```
    Receive L1;
A1: Set READ;
    Send L3;
    Receive L2;
    Unless INPUT A2;
    Set OUTPUT;
    Send L5;
    Go-to A1;
A2: Send L4;
    End;
```

Input Device

```
A1: Receive L3;
    If-internal-test A2;
    Set INPUT;
    Go-to A3;
A2: Set EOF;
A3: Send L2;
    Go-to A1;
    End;
```

Output Device

```
A1: Receive L5;
    Go-to A1;
    End;
```

A System of Processes described in

the Program Process Modeling Language

Figure 2-3 (Ref 11:220)

performance evaluation(Ref 8). The E-Net as developed was aimed at easing the modeling phase of system design and performance evaluation. It was also aimed at the simulation approach of analysis. To this end Nutt has made several modifications to the basic Petri net that enhance its utility in analysis of computer systems, especially in performance evaluation efforts. The three major modifications made are the resolution location used to resolve conflicts, the incorporation of time applied to transition firings, and the ability to have distinguishable tokens by adding attributes to the token definition. Due to these enhancements the E-Net is well suited for the task of modeling a communiction protocol and simulating its performance. The results of the simulation can be studied, either manually or by automated means to detect the presence of undesirable features and to evaluate the message transmission times. Due to these capabilities the E-Net was chosen as the underlying modeling technique. A summary of the important definitions pertaining to E-Nets follows. See reference 8 for a full description of E-Nets.

<u>Axiom 2.1</u>: The maximum number of locations connected to a transition is limited to four. The maximum number of transitions connected to a location is two, one directed into the location and one directed out of the transition. A <u>token</u> is a marker that indicates the status of a particular location.

<u>Definition 2.2</u>: A location is <u>empty</u> if it does not contain a

token, and full if it contains a token. The maximum number of tokens allowed to reside in a location is one.

Axiom 2.3: A location may only change status upon the action of one of the transitions it is connected to.

Axiom 2.4: The action of a transition is defined by providing the mapping whose domain and range are the status of locations connected to a transition. Status location are p-tuples where p is the number of location connected to the transition. If the status of a location is full the corresponding coordinate in the p-tuple is 1. If the status of a location is empty, the corresponding coordinate in the p-tuple is 0.

Example 2.5:

$$J(a,b,c): (1,1,0) \dashrightarrow (0,0,1)$$

Where J(a,b,c) is a function that describes the action of the "J" transition after it is enabled (Fig 2-4). The locations a and b are directed into the transition and c is directed out of the transition. The interpretation of this mapping is that if locations a and b are full and c is empty then the status´ of a and b change to empty and c changes to full. This is the only status triple that causes the transition to react.

Definition 2.6: A peripheral location is a location connected to exactly one transition. All locations that are not peripheral are inner locations.

Definition 2.7: A resolution location(r-location) is a location oriented into an X or Y transition, which when its

```
X(r,b1,b3,b4):    (0,1,0,0) --> (e,0,1,0)     r -->|--> b3
                  (0,1,0,1) --> (e,0,1,1)          |
                  (1,1,0,0) --> (e,0,0,1)          |
                  (1,1,1,0) --> (e,0,1,1)    b1 -->|--> b4

Y(r,b1,b2,b3):    (0,1,1,0) --> (e,0,1,1)     r -->|
                  (0,1,0,0) --> (e,0,0,1)          |
                  (0,0,1,0) --> (e,0,0,1)          |
                  (1,1,1,0) --> (e,1,0,1)    b1 -->|--> b3
                  (1,1,0,0) --> (e,1,0,1)          |
                  (1,0,1,0) --> (e,0,0,1)          |
                                             b2 -->|

                                                   |--> b3
F(b1,b3,b4):      (1,0,0) --> (0,1,1)              |
                                             b1 -->|
                                                   |--> b4

                                             b1 -->|
J(b1,b2,b3):      (1,1,0) --> (0,0,1)              |--> b3
                                                   |
                                             b2 -->|


T(b1,b3):         (1,0) --> (0,1)            b1 -->|--> b3
```

If r is a peripheral location, replace the character,e, by the undefined status character x; otherwise replace e by 0.

E-Net Transitions

Figure 2-4

status is empty or full, selects the alternate output location for an X transition, and may select the alternate input location for a Y transition. The existence of resolution locations is the key to the determinate action of the net. If a resolution location is a peripheral location it may be empty, full or undefined.

Unlike the Petri net where there is an unlimited number of transition types the E-Net allows only 5 types of transitions.

Definition 2.8: Given the set of locations, {b1,b2,b3,b4,r} where b1 and b2 are input locations, b3 and b4 are output locations, and r is a resolution location, define the five transition schemata, (types) of Figure 2-4.

For the graphical interpretation of Figure 2-4, a vertical line represents a transition, a resolution location is denoted by a hexagon, and all other locations are represented by a circle.

The process by which a transition moves from the status given in the left hand side of the definition to the status given in the right hand side of the definition is called the "firing of the transition".

Definition 2.9: A transition firing is a three phase operation consisting of the following phases:

1. Enable phase - A transition is enabled if the status of the locations connected to it satisfy the left hand side of the transition definition. The transition then begins operation.

2. Active phase - The transition action is in progress. The status of all locations does not change.

3. Terminate phase - The transition completes processing, changing the status of associated locations to agree with the right hand side of the transition definition.

Definition 2.10: A transition with the resolution location being a peripheral location is pseudo enabled whenever the status of the associated locations agree with the left hand side of the definition except for the undefined status of the resolution location. Pseudo enabled transitions can only become enabled transitions whenever the environment of the net defines the status of the resolution location as being either empty or full.

Definition 2.11: Given a transition, a, the transition time of a, denoted t(a), is the time required for the active phase of a firing of a.

Definition 2.12: An evaluation net structure, is a connected structure over the set of transition schemata of the net. An evaluation net structure, E, is denoted by

E = (L,P,R,A) where

L = A finite, non-empty set of locations.

P = A set of peripheral locations. $P \leq L$.

R = A set of resolution locations, $R \leq L$.

A = A finite, non-empty set of transitions, ai.

ai = (s,t(ai)), where s is the transition type and

t(ai) is the firing time for ai.

Example 2.13 An evaluation net structure

The following is a formal description of the evaluation net structure given in figure 2.5.

E = (L,P,R,A)

R = {r1}

P = {b1,b4}

L = {b2,b3} U P

A = {a1,a2}

a1 = (F(b1,b2,b3),t(a1))

a2 = (Y(r1,b2,b3,b4),(t0(a2),t1(a2)))

where the t(ai) are arbitrary transition time expressions.

```
                      a2
              r1  -->|
                     |
                     |--> b4
         a1          |
         |--> b2  -->|
    b1-->|           |
         |           |
         |--> b3  -->|
```

Figure 2.5

Definition 2.13: Given an evaluation net structure, E = (L,P,R,A), a *marking of E* is a function, M, taking L into the set {0,1,x}. The marking of E is denoted M(bi) = j for each bi in L and for some j in {0,1,x}. If M(bi) = 1, then bi is full. If M(bi) = 0, then bi is empty.

If M(bi) = x, then the status is undefined. The initial marking of E, denoted M0, is the marking of E which initially defines the status of L.

Definition 2.14: A simple token is a primitive marker which indicates that a location is full whenever it resides on that location. An attribute token is a unique simple token that has a finite, non-zero number of attributes associated with it.

Definition 2.15: A transition procedure describes the action of the environment and the associated locations of the transition on the token.

A transition procedure has the form

```
[p1 -> (e11;...;e1n):...:pk -> (ek1;...ekm)]
```

where the pi are predicates and the eij are expressions. A transition procedure is evaluated by the evaluating each pi in turn and executing the expression list of the first pi to evaluate to true. The transition procedure may alter the attributes of a token or the value of an environment variable.

Definition 2.16: An environment variable is an attribute token, K[n], where n > 0, that represents the status of a portion of the environment. An environment variable may be referenced in two distinct ways. It may appear as a left part of a transition procedure expression. The result, after evaluation, is the alteration of some attribute of the environment variable. An environment variable may also appear on the right side of a transition procedure expression or in a predicate. This kind of reference does not alter any attributes of the environment variable.

<u>Definition</u> <u>2.17</u>: A <u>resolution</u> <u>procedure</u> is an expression

```
R(r) = r:[<predicate> -> M(r) := s:
             <predicate> -> M(r) := 1 - s]
```

where s = {0,1} and r is the label of the peripheral resolution location.

A resolution procedure can only by evaluated whenever the associated transition is pseudo enabled.

<u>Summary</u>

Considerable amount of effort has begun to be expended in the area of computer communication protocol validation. The two primary areas of research are centered on graphical modeling techniques and modeling through high level languages.

The most readily adaptable methods to computer automation are the graphical techniques. Through the process of global state generation and reachability tree analysis the protocol can be evaluated to ascertain if any undesirable characteristics are present in the specification. A means of specifying the protocol has been developed in the form of the PPML which allows the protocol to be modeled in a natural way by Petri nets. The E-Net is an extension of the basic Petri net that enhances its applicability to the protocol evaluation effort.

III Requirements Definition

Introduction

The requirements definition phase of this project dealt with making decisions as to what services should be provided the user of the system and what the basic system requirements were in order to accomplish its primary purpose of protocol validation. There were two types of requirements considered. The first type was concerned with how the user would interface with the system. Issues considered during this part of the requirements definition were concerned with how to make the system easy to use. The second type of requirement was concerned with deciding the low level requirements dealing with how the system was going to fulfill its function of protocol validation. The results of these two requirement definition phases will be described in this chapter.

Overall Requirements

The first requirement is that the system should be easy to use. This would allow its use by people from many backgrounds, not necessarily computer networks. Ease of use required several other features be incorporated in order to accomplish this overall characteristic. The user should not be required to know a lot about how the system works in order to use it effectively. One way to insulate the user from the intimate details of the system is to have the system display options available to the designer at each level and allow a choice to be made from the input device. This form of interface is often associated with menus.

Another feature of user friendly systems is that changes are easy to make. The user should be able to selectively change any of the input parameters without having a major impact on the data entry phase. This means that the basic editing functions of deleting, inserting, adding, and changing should all be included in the input phase of the design.

Robustness is another feature associated with good system design. This means the system should be able to detect errors in input or specification and be able to recover from them without experiencing a fatal crash of the system.

Detailed Requirements

The basic requirement of the system is that it be able to validate protocols. Since computer networks communicate via the transmission of messages a requirement exists to be able to specify the message formats used by the communicating processes. Ideally, there should be no limitation as to the message types and formats. In order to accomodate this fact, the basic requirement is that a message format description have an associated type and any number of field descriptions.

The next issue considered in support of the validation requirement was that the protocol has to be specified in a manner that is conducive to the validation effort. This implies a means of specifying the protocol had to be developed or an already existing method had to be adopted. The second alternative was chosen. The PPML had been chosen as the means of specifying the protocol and the requirement now exists to provide a means to input the PPML description of the protocol. The basic requirements associated with this issue are that the protocol statements must be input, and a means of editing the PPML statements and the statement lists must be provided.

Since the method of validation had been decided upon, i.e. modeling, a requirement existed to provide a model of the protocol. This reduced to a requirment for a method to generate an E-Net from the protocol specified in the PPML. Along with this requirement was the requirement that a means of exercising the E-Net must be developed and a way to present the results for analysis had to be provided.

<u>Summary</u>

System requirements:

1) User Interface

   a) User friendly

      1. Use Menus for Input.

      2. Easy to edit input.

      3. Robust

         a. Detect syntactical errors in PPML.

         b. Detect semantical errors in PPML.

2) Validate Protocols

   a) Message Formats

      1. Many types of messages can be specified.

      2. Any number of message fields associated with a message.

      3. Message size will be represented.

      4. Field contents are variable.

   b) Protocol specified in PPML

      1. Any number of PPML statements in a process.

      2. Editing of PPML.

      3. Communication link characteristics specified.

         a. Channel capacity.

         b. Transmission rates of processes.

         c. Error and loss rates.

   c) E-Net built from PPML description.

   d) E-Net exercised.

      1. Manually

         a. User can examine present state of system.

         b. User can examine past states of system.

         c. E-Net exercised in response to user input.

      2. Automatic exercising.

         a. E-Net is exercised by machine.

   c) Output

      1. Token machine

      2. Automated analysis of Token Machine

      3. Message transmission statistics.

In summary , a means of inputing the message format has to be provided. A means of specifying the protocol in PPML has to be provided. A means of converting the PPML into an equivalent E-Net has to be developed and a means of exercising the E-Net and evaluating the results has to be designed.

This thesis effort will consist of designing the total system and implementing the PPML input, E-Net building and E-Net exercising phases. The next phase in project development is the design phase.

IV System Design

Introduction

The system design phase consisted of the overall design and the detailed design. The overall design dealt with the high level definition of user interface and system output. The design tool used in this phase was the SADT diagrams. Appendix B contains the system SADT diagrams. These diagrams give a global view of how the system is going to be developed to accomplish the goals of the requirements definition.

The detailed design stage consisted of further decomposition of the SADT diagrams into modules that could be implemented in a programming language. The design tool used during this stage was the structure chart. This turned out to be a very useful method. Using the structure charts facilitated several implementation issues and made the design process more manageable. These two observations can be demonstrated with examples. The fact that implemementation was made easier is a direct consequence of the form of the structure charts. Since parameters passed to procedures are incorporated in the structure charts the building of the procedure declarations followed naturally. The usefullness of structure charts as applied to design issues comes by way of the fact that when changes to modules were deemed necessary the impact of these changes could be ascertained by looking at the structure charts and determining which other modules would be affected by the change. The remainder of the chapter will elaborate on both the overall and detailed design issues.

Overall Design

The overall design phase of the project was mainly concerned with designing the highest level of system operation. This dealt primarily with designing the user interface to the system. As can be seen in the SADT diagrams in Appendix B the highest level is concerned with menu management. The main question that had to be answered during this phase was how the user was to access the features of the system through the menus. This led to having to decide how the user was going to get to the menu he needed.

There were two methods under consideration for the task of menu traversal. The first was to be able to go to any menu in the system from any other menu. The benefits of this scheme are obvious. It would be very easy and quick to get to the menu needed. There are some drawbacks, though. First, every menu would have to reference every other menu in the system. This entails having very large item lists in the menus. Secondly, when thinking about program maintenance as well as development this scheme presents a formidable challenge. All menus would have to reflect any new menus added or deleted from the system. Due primarily to the first problem noted, this scheme was rejected in favor of the next method.

The menu traversal method chosen allows the user to step through the menus one level at a time. This is very similar to a tree traversal where it is necessary to go to a higher level node in order to get to another part of the tree at the same level as the present node. The means of achieving the single level step system is to have an item labeled 'EXIT' in each menu that can take you to the next higher level menu. The benefits of this method are that the menus are smaller and it presents a more structured appearance as well as being more structured in its operation. The drawbacks are that it takes longer to get from low level menus to higher level ones and back down.

The second major issue of the overall design was what menus would be needed at the top level. The menus decided upon reflect the overall structure of the system. There are three main systems and correspondingly three main menus. The first is the Message Format Menu(Appendix B). From this menu the user can select several options in order to edit the message format list. The second option availlable at the highest level is the PPML Input Menu. This menu has options to edit the Global variable list, the PPML statement lists and to input the number of communication links. The third option that can be chosen by the user at the top level is the Evaluation menu. From here the system builds the E-Net and exercises it presenting the results to the user in several forms for analysis as described in the next section.

The first characteristic of messages that became apparent was that each message processed by the system had an associated identification. This usually manifested itself in the form of a dedicated field that contained information as to the message type. An example of this feature can be seen in the messages used by the LSINET where each message has a three byte field that identifies the particular message type. Typical contents of this field are 'REQ' or 'COM'. The REQ message is sent by a remote terminal to either the spooler for output servicing or the mass storage system for file transfer. The COM message is sent from one remote to another to set up a conversational session between the two terminals. The format of a typical LSINET message is shown in Figure 4-1.

Another characteristic of messages is that they are composed of one or more fields. There may be only one field as in a message used solely for the purpose of acknowledgement with no other information contained in the message or the message may be composed of several fields of which any number of the fields may have information pertaining to the protocol aspects of the system. In regard to this characteristic one can again look at the LSINET where a message contains several fields. Each message traveling through the net has a header with routing information and several fields used by the communication computers to set up the communication process and send data

Request to Spool Message

Char Number	Field description
0	ASCII STX character
1-3	Three character message type, REQ
4-6	System ID at destination computer, SPO
7-9	System ID at source computer
10 - 23	Filename
10-11	Disk ID ,e.g. "1:" or "A:"
12-23	Filename and extension
24	Priority 0-9 in ASCII
25	Destination Device in ASCII
26 - 30	Filesize in 64 word blocks
31	ASCII ETX character

LSINET Message Format

Figure 4-1

to each other. By examining Figure 4-1 we see that routing information is contained in characters 4-6 while the rest of the message contains information used by the communicating computers to effect the requested service. There is in theory no limit to the number of fields making up a message.

The third main characteristic of messages is that they contain information. The information may be used by the communication subnet for routing purposes or it may be used at the host computers to effect the information transfer function of the network. In general, messages contain both types of information. The way the information is represented in the message varies. It may be in binary form, ASCII character form or BCD integer or any other form agreed on by the communication processes. The type of information used by the protocol system is usually referred to as ´control´ information. Typical uses for the control information are to identify the sending computer, identify the type of message, acknowledge a particular message by its sequence number and so on. The use of the information is limited only by the designer´s imagination. As described in Chapter 3, the information is represented by integers in the validation system.

The fourth main characteristic of messages their size The size is represented by the number of bits making up the message. This too is a variable feature of messages.

It appears as though there are four main characteristics of messages that are required in order to adequately model the messages used by the system. They are:

1) Message type
2) Variable number of fields
3) Information in several forms
    a) Character data
    b) Bit stream data
    c) Integer data
4) Message size

These are the features of messages that the message format input system will ask for. The actual means of input will be described in the next section.

Message Format Input

Now that the decision had been made as to the information that had to be collected about messages, the means of input could be designed. The user enters the message format input menu(Appendix A-2) by responding to the Master menu(Appendix A-1) with a '1'. The user can access several message format editing functions once at the Master Message Format Menu. These include entering the format for a message that has not been previously described, changing the description of a message, deleting a message from the list or listing the information on the messages on the message format list. After selecting the desired function, the user is prompted for the remainder of the information. Once all the messages have been described the user can return to the system master menu by choosing option '5' from the Master Message Format menu.

PPML Input

Prior to actually designing the PPML input method it was necessary to decide what information was had to be collected during this phase. Since this is the last phase before building the E-Net all the remaining data needed to build the net had to be input. The remaining data was the global variable definition, the number of communication links and the actual PPML statements. Accordingly, the means to input these three items is imbedded within the PPML input menu.

In order to specify the number of communication links in the protocol system the user selects option '7' from the master PPML menu. A prompt is then displayed and the data can be entered. This information is used later while building the E-Net. After entering the data the master PPML menu is once again displayed.

The user can enter the global variable input menu (Appendix A-4) from the master PPML menu by responding to the prompt with a '1'. Again, before designing the global variable input menu an analysis of global variable attributes had to accomplished to decide what information had to be input. There are two characteristics of global variables that impact the design of the method of entry. The first is that global variables have names. The name of a global variable is how the PPML statements reference the variable. The second attribute specifies the value of the global variable. As described in Chapter 3, the only kind of value the global variables assume is integer.

Once at the global variable input menu (Apppendix A-4) the user can enter the variable name and initial value. The system will prompt for the required data. After the global variables have been entered a selection of '4' at the global variable menu will return the user to the Master PPML input menu.

PPML Input

The only remaining information to be input from the master PPML menu is the actual PPML statements. Before developing the specific method of entry a decision had to be made as to the form of the PPML statements. The question of syntax was considered first. Two syntaxes were considered. The first was an unstructured syntax where the keywords and fields of the statements would be seperated by some kind of termination character such as a space and there could be any number of them between fields. This form would be representative of many of the present day higher level programming languages like Pascal or 'C'. Due primarilly to the complexity of the process required to 'compile' this form of input to the E-Net structure. The unstructured syntax was rejected in favor of a structred one.

The syntax of the PPML statements is considered structured due to the fact that the different fields of the statements have to begin in particular columns in the statement line. Each statement is composed of three fields;

the label, the statement type and the operand field. The label must start in column 1 of the line and can contain up to 10 characters. The characters can be any recognizable ASCII character to include upper and lower case letters, digits or special characters such as ´&´,´%´,´[´,etc. The statement type field has the most restrictions. It begins in column 12 and is 10 spaces wide. This field contains the basic PPML statement type. The characters in this field can only be the capitalized statement type and must be one of the following:

1) SET
2) SEND
3) RECEIVE
4) IF
5) GOTO
6) UNLESS
7) ASSIGN
8) END

The above list reflects several modifications to the PPML format as described by Riddle(Ref 12). The two primary changes are the modification of the If-Internal-test to the generic IF and the inclusion of a new statement type ´ASSIGN´. The IF statement is now capable of recognizing a more general predicate than the IF-Internal-Test described by Riddle. The form of the predicate is as follows:

```
PRED ::= { X relop Y }
```

where X,Y,and Z are field names or global variables or constants and ´relop´ is one of (´<´,´>´,´=´) . This set of operators was chosen to keep the set small and because all other relational operations can be built using these. The modification to the basic PPML were made to accomodate the features of the E-Net. Since the E-Net has the capability of making decisions based on the value of some variable, such message field or global variable the IF statement was included to allow access to this feature. Along this same line the E-Net has the capability to manipulate varibles. Since the PPML description given by Riddle does not incorporate this capability a new statement had to be incorporated in the PPML. The ASSIGN statement resulted.

This brings us to the point of having to decide on how to represent global variables and field names in the PPML statements. It was decided to have field names enclosed in square brackets ,´[´ and ´]´, and to allow the global variables to be referenced simply by their names. There is no message qualification necessary to identify the fields because a PPML statement can only reference the ´current´ message. This is the message that resides in the input place to the transition representing the statement. This will be discussed further in the next chapter. A typical example of global variable and field name reference in a PPML statement is :

```
          IF         { [FIELD1] = SEQ } THEN LABEL1
          ^          ^
 column#> 12         23
```

Where 'FIELD1' is one of the fields of the current message that was described in the Message Input section and 'SEQ' is a global variable. As can be seen by the above example, the predicate must be enclosed in curly brackets and the predicate is followed by the keyword 'THEN' which is in turn followed by a label to go to if the predicate evaluates to true. More examples will be shown in the next section.

The operand field contains the remainder of the PPML statement. It starts in column 23 of the statement line. The contents of the operand field vary according to the PPML statement type and the variables accessed.

Detailed PPML Syntax Description

The syntax of each statement type will now be described. The basic structure of each statement as described in the previous section can be depicted as:

```
| LABEL    ||STMT TYPE|| OPERANDS...........................
^          ^          ^
1..........12.........23.....................................
```

The descriptions can be interpreted as follows. The first line is a column indicator. The '1' in the top line indicates that the LABEL field starts in column one. The '12' indicates that the PPML statement field starts in column 12. The '23' indicates that the remainder of the statement begins in column 23.

1) <u>SET</u>

```
1..........12.........23...............................
LABEL1     SET        { M1 }
```

The SET instruction specifies a particular type of message to be built. The message type must have been previously defined in the Message Format input phase.

2) <u>SEND</u>

```
1..........12.........23.......................................
LABEL2     SEND       { LINK1 }
```

LINK1 is the name of one of the communication links. The specific data pertaining to the links will be input during the E-Net building phase.

3) <u>RECEIVE</u>

```
1..........12.........23.......................................
LABEL3     RECEIVE    { LINK2 }
```

This statement specifies that the input buffer associated with communication link LINK2 should be accessed and any message there should be retrieved.

4) <u>IF</u>

```
1..........12.........23.......................................
LABEL4     IF         { [SEQ] = 5 } THEN LABEL17
```

where SEQ is a field name ;

```
LABEL5     IF         { SOURCE = 3 } THEN LABEL23
```

where SOURCE is global variable;

```
LABEL6     IF         { [FIELD2] = GLOBAL1 }
```

where FIELD2 is a field name and GLOBAL1 is a global variable.

The left and right sides of the predicate can not be compound expressions. It can only be a field name enclosed in square brackets or a global variable or a constant.

5) GOTO

```
1..........12.........23.......................................
LABEL 7    GOTO       START
```

The GOTO instruction is one that does not require the curly brackets around the operand field. This instruction will transfer control to the statement in the same process with the label START.

6) UNLESS

```
1..........12.........23.......................................
           UNLESS     { M1 }  LABEL25
```

This statement was included because it was included in the list of basic statements as described by Riddle. It could be replaced by the more general IF-THEN statement. This statement will transfer control to LABEL25 unless the current statement type is M1.

7)ASSIGN

```
1..........12.........23.......................................
LABEL45    ASSIGN     { [SEQ] = WINDOWBOT + 1 }
```

Where SEQ is a field in the current message and WINDOWBOT is a global variable. The effect of this instruction is to assign the value of the right side to the variable on the left side. The basic form of the ASSIGN statement is:

```
1..........12.........23.......................................
LABEL      ASSIGN     { X = Y OP Z }
```

Where X is a variable(field or global), Y and Z are variables(field or global) or constants. The OP is one of

(´+´, ´-´ , ´*´). The variable on the left side can appear on the right side. In which case the right side is evaluated with the right side value existing before replacement. This is the same as the Pascal assignment statement.

8) <u>END</u>

```
1..........12.........23.......................................
           END
```

This statement identifies the end of the process description.

Figure 4-2 is the system described in Figure 2.3 using the syntax from above. Figure 4-2 depicts the PPML description of three seperate communicating processes. The processes are named ´Program´, ´Input Device´, and ´Output Device´. The action of the system is initiated by Program receiving a message from link L1. Program then sends a READ message to Input. The Input process then returns either an INPUT or EOF message back to Program. If program receives an INPUT then an OUTPUT message is sent to Output Device and program waits for another message from link L1. If an EOF message is received by Program then Program is terminated and the system has completed its function as indicated by all processes reaching their respective end statements.

Now that the syntax had been decided on the problem of inputting the statements into the system could be attacked. The user can enter statements by selecting option number ´2´ in the Master PPML menu. This is used to enter the PPML statements for the first time. That is when there are no statements already assigned to a process. When selecting the new PPML statement option the user is requested to enter the process number of the process that the new statements apply to. The validation system references processes by number and not by name as in the PPML description of Figure 4-2. One of the first things the program does when activated is to ask for the number of communicating processes. In our example of Figure 4-2 the answer would be 3. This information is stored and used in the PPML input as well as the E-Net building phases. So, after the user responds with a valid process number a prompt of the form

```
| LABEL  ||STMT TYPE|| Remainder of statement
==============================================================
```

is displayed and the PPML statements are then input to the system. After the last statement('END') has been entered, the user must enter a single 'Z' character to indicate he has finished entering the statements for that process. At that point he is returned to the master PPML input menu. He can then enter the statements for any other processes or delete, insert or change statements to any process.

Protocol Evaluation

Once the message formats and PPML statements have been input the protocol evaluation menu can be entered. It is from here that the E-Net is built and exercised. The first thing this menu accomplishes is requesting the final piece

Program

```
1..........12.........23..................
           RECEIVE    { L1 }
A1         SET        { READ }
           SEND       { L3 }
           RECEIVE    { L2 }
           UNLESS     { INPUT } A2
           SET        { OUTPUT }
           SEND       { L5 }
           GOTO       A1
A2         SEND       { L4 }
           END
```

Input Device

```
1..........12.........23........................
A1         RECEIVE    { L3 }
           IF         { X = Y } THEN A2
           SET        { INPUT }
           GOTO       A3
A2         SET        { EOF }
A3         SEND       { L2 }
           GOTO       A1
           END
```

Output Device

```
1..........12.........23.........................
A1         RECEIVE    { L5 }
           GOTO       A1
           END
```

A System of Processes described

in the Modified Program Process Modeling Language

Figure 4-2

of information needed to build the E-Net, the link data. Link information is collected for the number of links specified in the PPML menu. The data collected is the link name and the number of messages allowed on the link at one time. This number will be a function of the buffer size of the sending and receiving processes and the channel capacity of the link. After this information has been input for all the links the system will build the E-Net. The design of the means of transforming the PPML specification into a representative E-Net consumed the bulk of the design effort and a description of the design process and results follows.

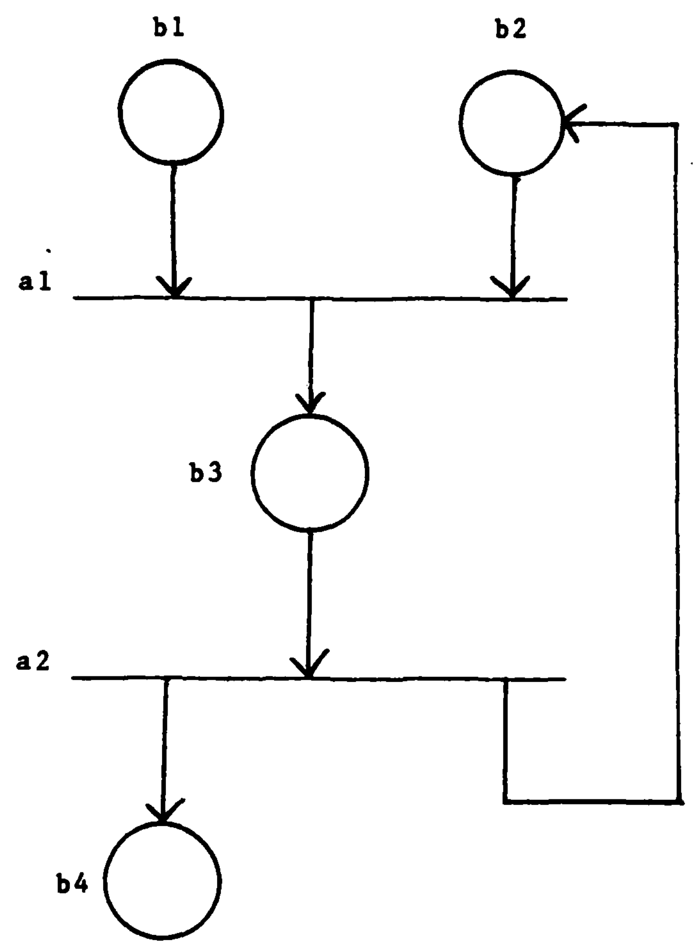
PPML to E-Net Process Design

The design of the E-Net building process proceeded in two phases. The first was the design of the E-Net primitives that would be used to model the PPML statements and the second phase dealt with how to link the E-Net primitives together to construct a complete net. The design of the primitives will be described first followed by a description of the linking process.

Since the underlying structure of the E-Net was already defined as transitions and places the emphasis of this phase was on how to model the PPML statements using the E-Net constructs of places and transtitons. Each PPML statement type was analyzed to determine the actions of the statement with regard to the communication process. The results of this analysis follows.

1) SET - The SET statement accesses the message format list and builds a token that represents the type of message prescribed by the SET instruction. Figure 4-3 depicts the graphical and formal descriptions of the SET instruction. A summary of the operation of this primitive follows. When the net is initialized the marking of b2 is set to the token representing the message type prescribed by the SET statement. The set instruction is enabled when a token is placed on b1. The transition procedure of a1 specifies that the token at b2 is transfered to b3 while the markings of b1 and b2 become empty. This action enables transition a2. When a2 fires, the token representing the message is transfered to b4 which indicates the completion of the SET instruction and a copy of the message format is also sent to b2 so that is available for the next invocation of the SET instruction. Place b4 will be linked to the primitive of the next sequential PPML statement. The time associated with this primitive is contained in a2 with the firing time of a1 being 0.

2) SEND - The SEND instruction has to transfer the token in its input place to the link process and to the enabling place of the next instruction. This is achieved by using a ´F´ transition. A representation of the SEND primitive is shown in Figure 4-4. The firing time associated with this transition is 0. The link process will take care of the transmission time of the message. The link primitive will be described later.



a1: (J(b1[x], b2[x], b3[x]), 0 seconds,
[T -> (M(b3[x] := M(b2[x]))])

a2: (F(b3[x],b4[x],b2[x]), S seconds)

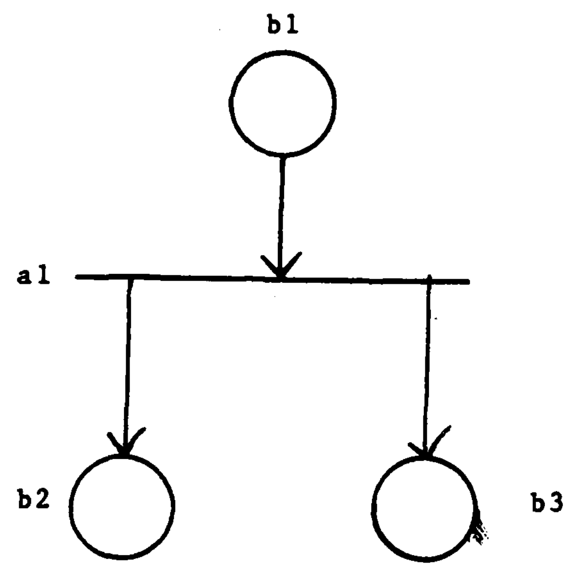
where 'x' is the number of fields associated with the message being built and 'S' is the execution time of the SET instruction.

E-Net Construct For PPML Statement

SET

Figure 4-3

3) RECEIVE - The RECEIVE primitive turns out to be the most complex of all the constructs. The complication arises due to the fact that the RECEIVE construct chosen not wait for a message. If there is no message the PPML process continues execution. If it did wait the statement could be modeled using a simple 'J' type transition. On the surface it looks like a 'Y' transition would apply. But when considering the firing scheme of the 'Y' transition a problem arises. If there is a token in the receive buffer then we want that token to be passed to the next transition and have the token in the enabling transition to be discarded. The 'Y' transition does not model this activity. When a 'Y' transition fires with both input places full, one of the input places becomes empty and the other remains full. Figure 4-5 is the primitive that models the RECEIVE action that we want. The primitive is activated by placing a token in b1. The resolution procedure for r1 is then evaluated. The resolution procedure sets r1 to 1 if there is a token in b3, the receive buffer. If there is a token in b3, then the transition procecdure of a1 discards the enabling token by passing it to b5 while at the same time setting b11 so that a3 can be enabled. If there is no token in b3 then the enabling token is passed to b4 and b11 is again set to enable a3. At the completion of the a1 firing either b3 will have a token from the link or b4 will have a token from a1. In any case, a3 will now be enabled. This occurs because by setting b11 when the resolution



a1: (F(b1[x],b2[x],b3[x]),0)

<u>E-Net Construct For PPML Statement</u>

<u>SEND</u>

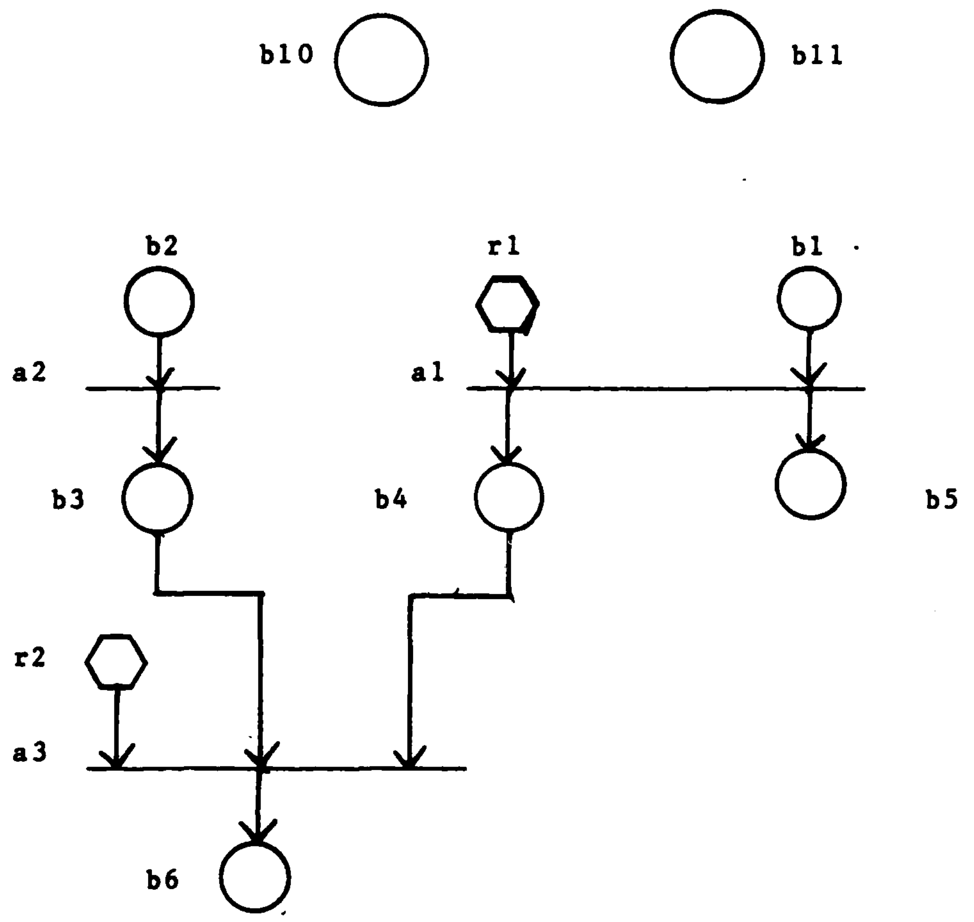
<u>Figure 4-4</u>

procedure for r2 is evaluated it will evaluate to a 0. The transition firing of a3 will then take the token from the one input place that has the token and pass it to b6. The way a1 knows the status of the receive buffer is by b10. Place b10 is an environment variable used to indicate that there is a token in b3. The value of b10 is set by the link process at a2.

4) <u>ASSIGN</u> - The ASSIGN statement emulates a Pascal assignment statement. The value of the variable on the left side of the assignment is replaced with the value computed from the expression on the right side. As previously noted the expression can have at most one arithmetic operator with at most two operands. This may seem like quite a restriction but when considering the operations performed by the control portions of a network this limitation does not create a severe problem at all. The ASSIGN statement is modeled by a simple 'T' transition with the transition procedure specifying the actual assignment. Figure 4-6 is a representation of the model of the ASSIGN statement.

There is a special form of the ASSIGN statement that allows the user to keep track of when a message is received. This is needed so that the time a message is received at its final destination can be recorded. The format of this statement is:

```
1..........12..........23................................
           ASSIGN     { [TIMERCVD] = CLOCK }
```



```
a1: ( X(r1,b1[x],b4[x],b5[x]),(0,0),

      [ M(r1) = 1 -> (M(b5) := 1);
                     (M(b10) := 0 ):
        T -> ( M[b4] := M[b1] ) ])

a2: ( T(b2[x],b3[x]), (T seconds ),

      [ T -> M(b10) := 1 ])

a3: ( Y(r2,b3[x],b4[x],b6[x]), (0,0),
        [-] )

r1: [ M(b10) = 1 -> M(r1) := 1 ]

r2: [ M(b11) = 1 -> M(r3) := 0 ]
```

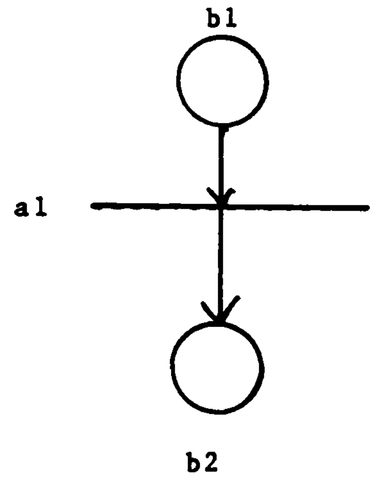
RECEIVE

Figure 4-5

The designer uses this statement to set the TIMERCVD attribute of the message just received.

5) IF - The IF statement allows the sequencing of PPML instructions to take on of two aternative paths depending on the value of some system variable. The essence of the IF statement is embodied in the resolution procedure associated with the resolution place of the 'X' transition used to model the IF statement. Figure 4-7 shows the IF primitive and an example of its use. There is one point that should be brought out at this time concerning a convention implemented to make the branching calculations standard within the system. As can be seen from Figure 4-7, when the branch is taken the token is placed in the output place corresponding a '1' in the resolution place. This is standard for all branch instructions, specifically the 'IF', 'UNLESS', and 'GOTO' PPML statements.

6) UNLESS - The UNLESS statement is very similar to the IF statement and as one would suspect the structure of the UNLESS primitive is very similar to IF primitive. The only difference between the two is that the UNLESS resolution procedure does not have to find the appropriate attribute as the IF statement does. It knows that the message type is always in attribute number one. Again, this is another convention used to simplify and standardize the primitives. Figure 4-8 is a depiction of the UNLESS primitive along with an example.



```
al: ( T(bl[x],b2[x]),(ASSIGNtime),
       [ as specified in PPML statement] )
```

example:

If the PPML statement is :

```
            ASSIGN            { [F1] = SEQ + 2 }
```

then the transition procedure would be:

```
   [ T -> (bl[y] := M(bZ) + 2 ]
```

where y is the attribute number of the token(message) in bl that corresponds to the field ´F1´ and bZ is the place that contains the envitonment variable corresponding to the global variable named ´SEQ´.

E-Net Construct For PPML Statement

ASSIGN

Figure 4-6

7) GOTO - The GOTO is an unconditonal jump to the target label specified in the PPML statement. Figure 4-9 shows how the GOTO is modeled.

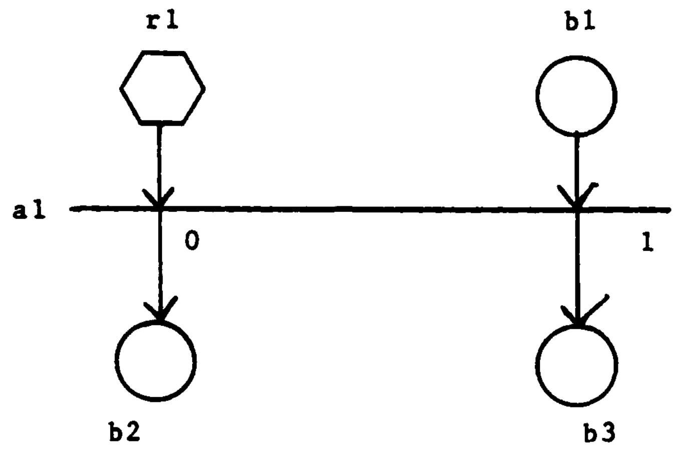
8) END - The END statement has no associated transition. When an END statement is reached the process is no longer active and this is modeled by not having any enabled transitions in the process.

Primitive Linking

Now that the primitives had been designed a scheme to connect them together had to be developed. There were three different types of constructs that had to be considered in this effort. By looking at the PPML descriptions of a process one can detect the three linking forms of sequential, branch, and link. The sequential form arises due to the underlying sequential nature of the PPML description and this form merely connects one primitive to its predecessor in the PPML description. The branch type of interconnection arises due to the branching PPML statements, IF, UNLESS, and GOTO. The link type of interconnection relates to the SET and RECEIVE PPML statements. Each of these will be described in this section.

Sequential Primitive Linking

The basic type of interconnection in the E-Net building process is the linking of one primitive construct to the primitve of the next PPML statement in the PPML description of the process. This is represented in Figure 4-10. This



Where b3 is the input place to the transition identified by the target label in the PPML statement.

```
a1: (X(r1,b1[x],b2[x],b3[x]),(IFtime),
     [] )

r1: ( [ depends on predicate ])
```

example:

if the PPML statement is:

```
           IF             { [F1] = 3 } THEN  LABEL7
```

then the resolution procedure for r1 would be:

```
    [ M(b1[y] = 3 -> M(r1) := 1;
      T -> M(r1) := 0 ]
```

E-Net Construct For PPML Statement

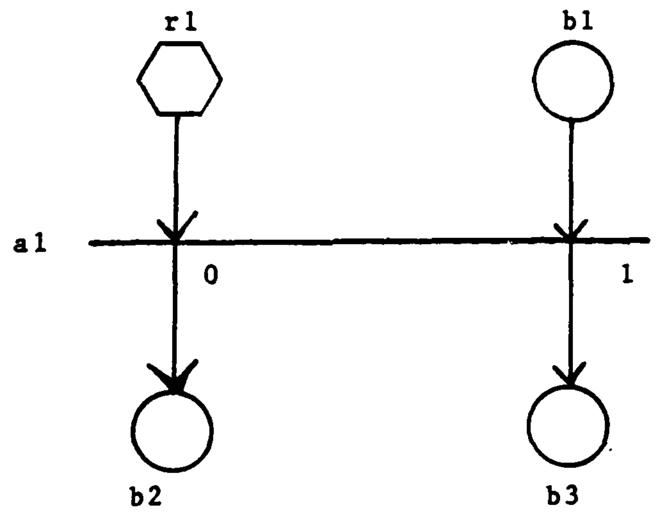
IF

Figure 4-7

type of linking takes place as each PPML statement is being processed. When a statement has been processed and the primitve linked to the previous primitive there is one output place that is used to indicate the completion of the execution phase of the primitive. Place b2 in Figure 4-10a is the execution completion indicator for the primitive modeling the SET instruction. Figure 4-10b shows the situation after the ASSIGN instrucion has been processed. Place b2 has become the input place that enables the ASSIGN primitive and b3 is the execution completion indicator for the ASSIGN primitive. This type of linking applies to all primitives except the GOTO. If the previous PPML statement is a GOTO then the place that represents the completion of the GOTO is not linked to the next sequential PPML primitive. The GOTO primitive has to be linked to its target primitive. This will be described in the next section.

Branch Primitive Linking

Since the branch PPML instruction,e.g. IF, UNLESS,and GOTO, are used to break out of the sequential execution of the process the linking of the primitves must reflect this purpose. The IF and UNLESS have to be linked to the next sequential primitive as well as the primitve that represents the instruction associated with the target label. The GOTO has to be linked only to its target primitive. This form of primitive linking would not present any difficulty if it was possible to allow multiple input transitions to a place.



Where b3 goes to the transition identified by the target label of the statement.

```
a1: ( X(r1,b1[x],b2[x],b3[x]),(UNLESStime),
      [] )

r1: [ depends on PPML statement ]
```

example:

if the PPML statement is:

```
        UNLESS               { M1 } LABEL12
```

then the resolution procedure for r1 would be

```
     [ M(b1[1] = 'M1' -> M(r1) := 0;
       T -> M(r1) := 1 ]
```

Where attribute number one contains the message type.

E-Net Construct For PPML Statement

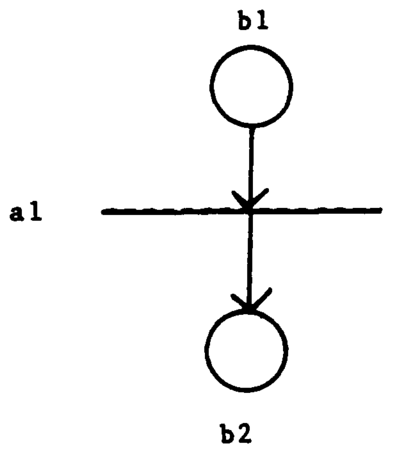
UNLESS

Figure 4-8

The Petri net would allow this but the E-Net does not. So, a more complicated interconnection scheme results. The problem arises when a branch instruction references a primitive that has already been linked to its predecessor. This situation is depicted in Figure 4-11a. The question is how to let a primitive be the successor of more than one primitive. The solution to this problem is shown in Figure 4-11b. An auxiliary 'Y' trnsition is used to buffer the two input transitions to the the target primitive. This construct works as follows. If we specify that the resolution procedure for r1 always returns a one then the auxiliary transition az will become enabled whenever either of its input transitons becomes full. This enabling token will then be passed on to bx thereby enabling a1.

Communication Link Primitive Linking

The last construct in the PPML description that had to be modeled was the link process. Since the communication link to be modeled is one in which the transmission media is directly connected to the processes accessing the link we can use a queue construct to connect the two processes. Figure 4-15 shows the resulting primitive that models a communication link. All transitions have a firing time of zero except for the last one which is called the 'Linkend transition'. All information regarding the characteristics of the transmission media is contained in the 'Linkend transition. The information contained there includes the transmission time on the link, the error rate and message



Where b2 is the input place of the transition associated with the target label in the PPML statement.

```
a1: ( T(b1[x],b2[x]),(GOTOtime),
      [] )
```

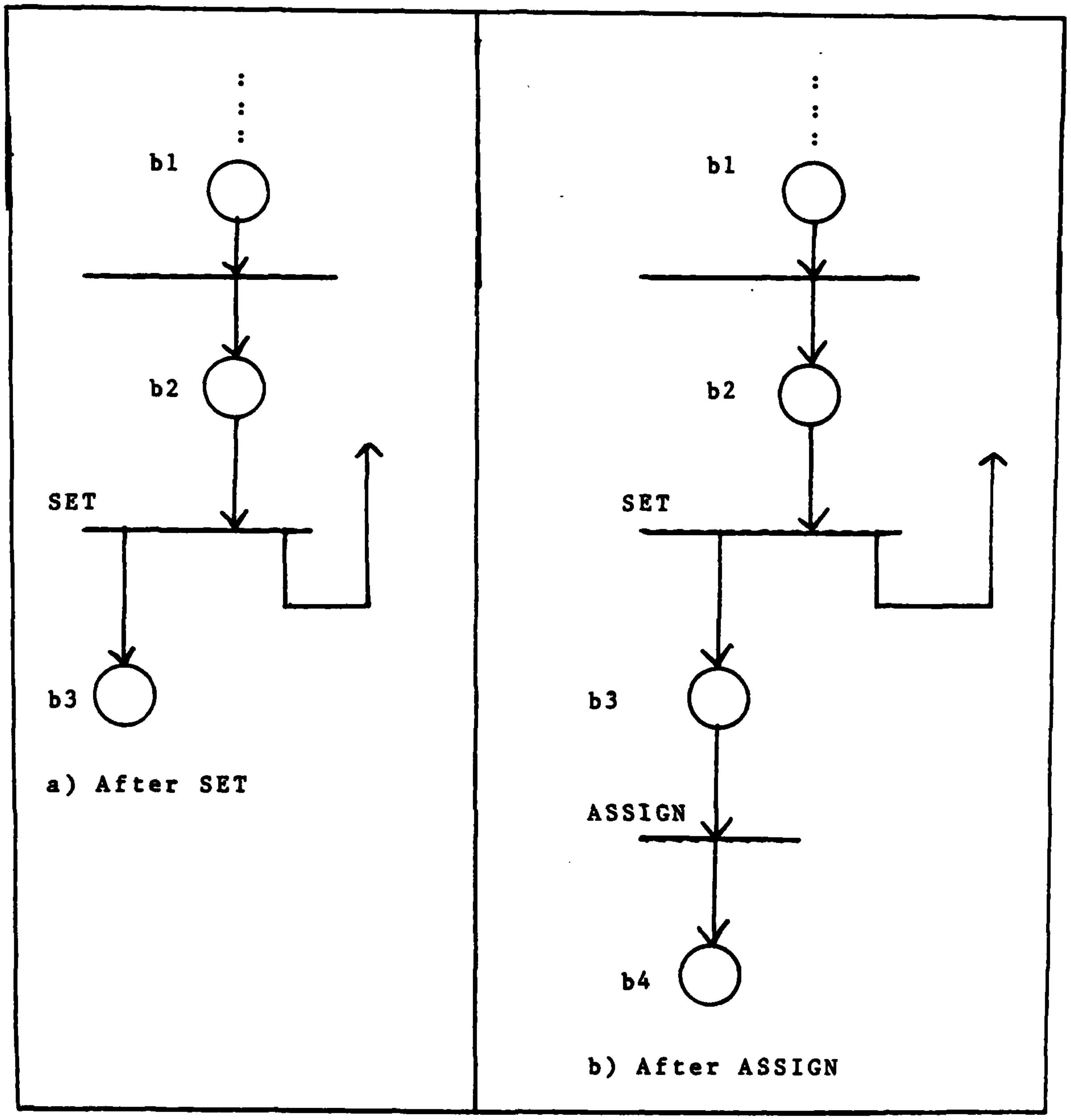
E-Net Construct For PPML Statement

GOTO

Figure 4-9

loss rate. The transmission time includes the time to put the message on the link by the SEND instruction, the time to send a bit along the link and the time to receive the message at the receiver. The SEND time and RECEIVE times are a function of the baud rate of the processes. The action of the link can be described as follows. When SEND delivers a token to the link the token will propagate to the last empty place on the link before the 'Linkend' transition. If the output place of the 'Linkend' is empty then the linkend transition will become enabled. If the output place to the linkend is full then the linkend will not become enabled until its output place becomes empty through the action of a RECEIVE primitive. When a token is removed from the linkend output place the input place to this transition is examined to see if the transition should once again become enabled.

The communication link constructs are connected to processes via the SEND and RECEIVE PPML statements. The function of the SEND statement in relation to the communication link is to add another token to the end of the link construct while the RECEIVE function is to remove a token from the link process. This would be trivial to model with E-Net constructs if we limited the definition of the processes to prohibit more than one SEND or RECEIVE to access the same link from the same process. This clearly is not a viable alternative. There are in all probability going to be multiple SEND and RECEIVE statements referencing



```
PPML process:          :
                       :
                      SET           { M1 }
                      ASSIGN        { X = Y }
                       :
```

<u>Example of</u>

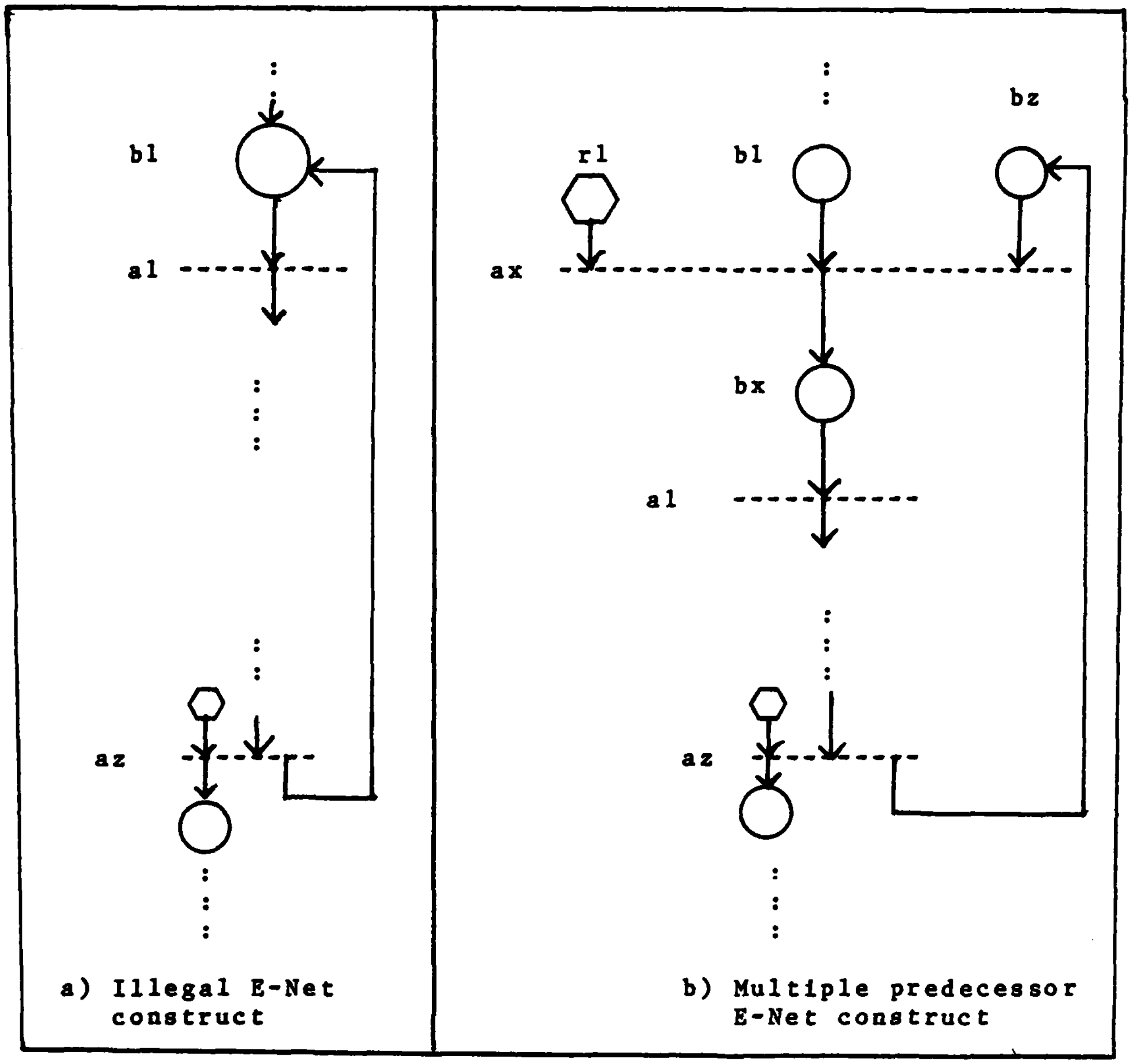
<u>Sequential Primitive Linking</u>

<u>Figure 4-10</u>

a link This problem is similar to the one described in the previous section. We want multiple primitives to be linked to the communication link constructs. Indeed the SEND situation is identical to the aforementioned problem and can be solved in the same manner. Figure 4-12 depicts this solution for the SEND statement.

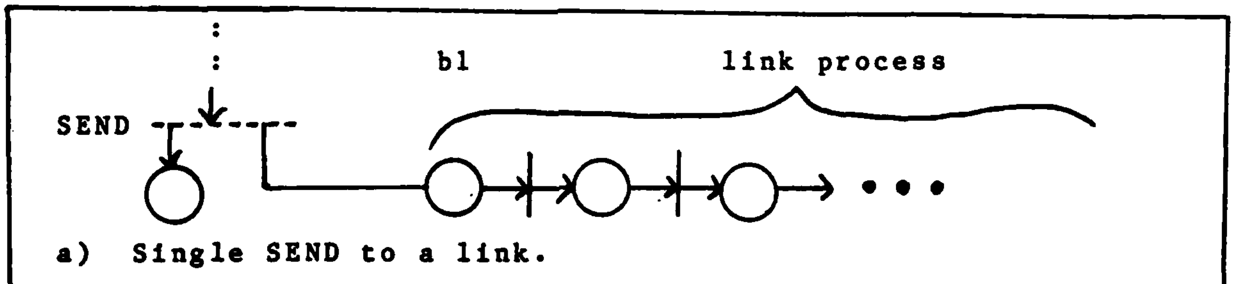
Unfortunately, the RECEIVE presents a more difficult problem. This arises due to the fact that a token has to be delivered to a particular requesting transition. The situation is the converse of the SEND problem. Instead of having multiple predecessors to a transition we have to model multiple successors to a transtion. The place representing the end of the link process has to be connected to all RECEIVE primitives that access that link. We have the problem shown in Figure 4-13. Since a place can have at most one output transition in an E-Net, a means had to be devised to allow a place to be connected to many transitions.

The solution to this problem is represented by Figure 4-14. The transitions between the link and the RECEIVE primitives are steering transitions. Their purpose is to propagate the token from the link to the apropriate RECEIVE transition. The way they do this is by having the resolution procedures reference global variables set by the receiving transition.

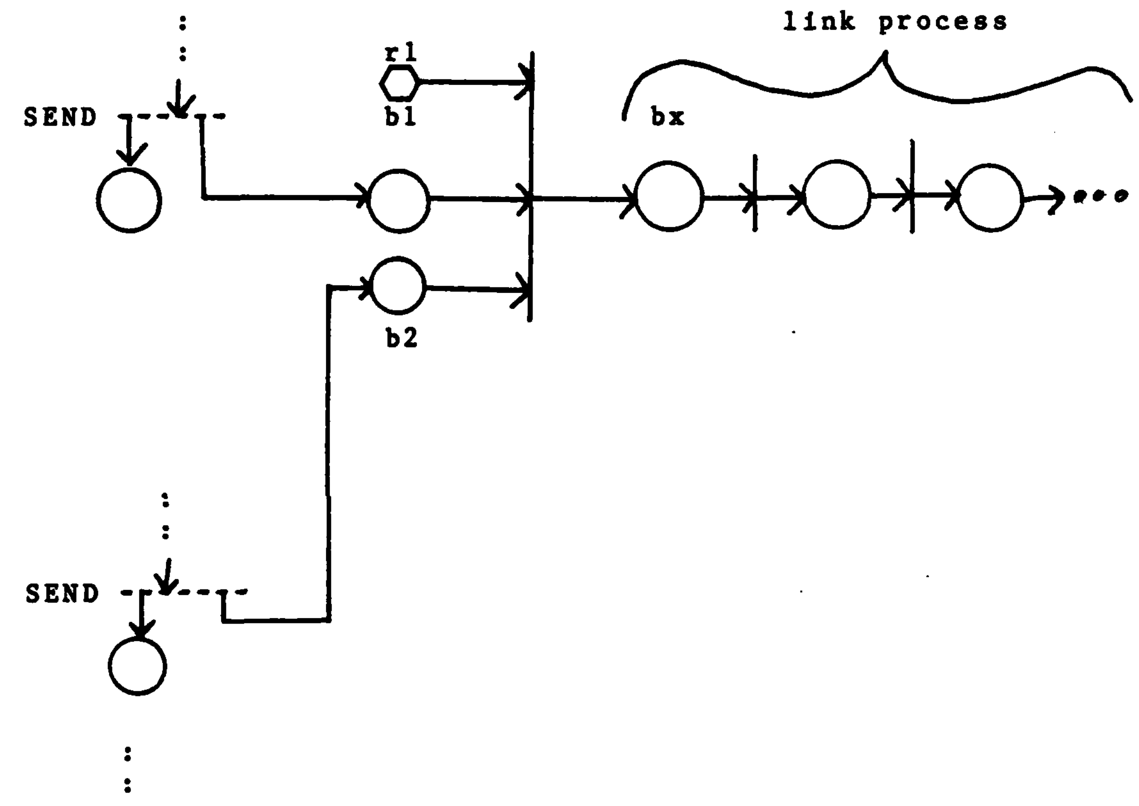

bl
al
az
a) Illegal E-Net construct
rl
bl
bz
ax
bx
al
az
b) Multiple predecessor E-Net construct

```
PPML PROCESS:
                              :
                              :
LABEL1          ASSIGN              { X = Y }
                              :
                              :



                              :
                              :
                IF                  { X = Y } THEN LABEL1
                              :
                              :
```

<u>Multiple Predecessor Construct</u>
<u>Figure 4-11</u>



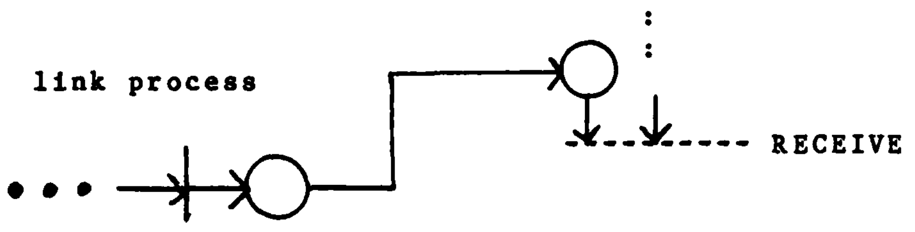
a) Single SEND to a link.



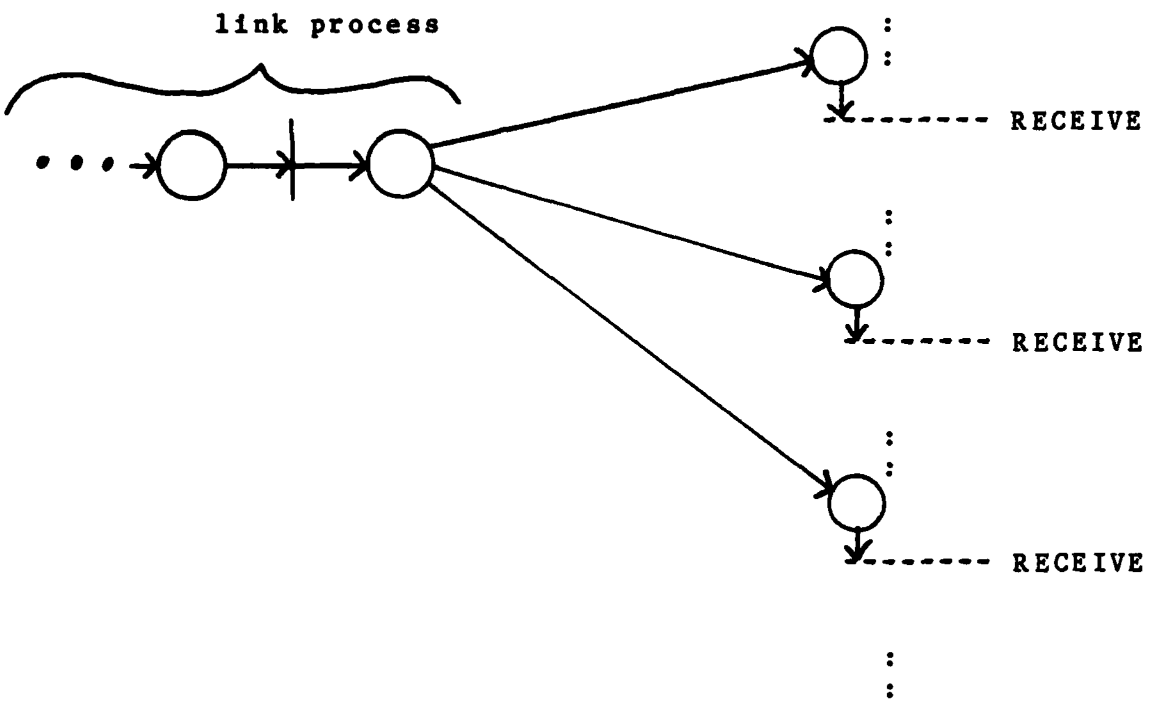
b) Multiple SEND statements to same link.

Multiple SEND Statements to Same Link

Figure 4-12



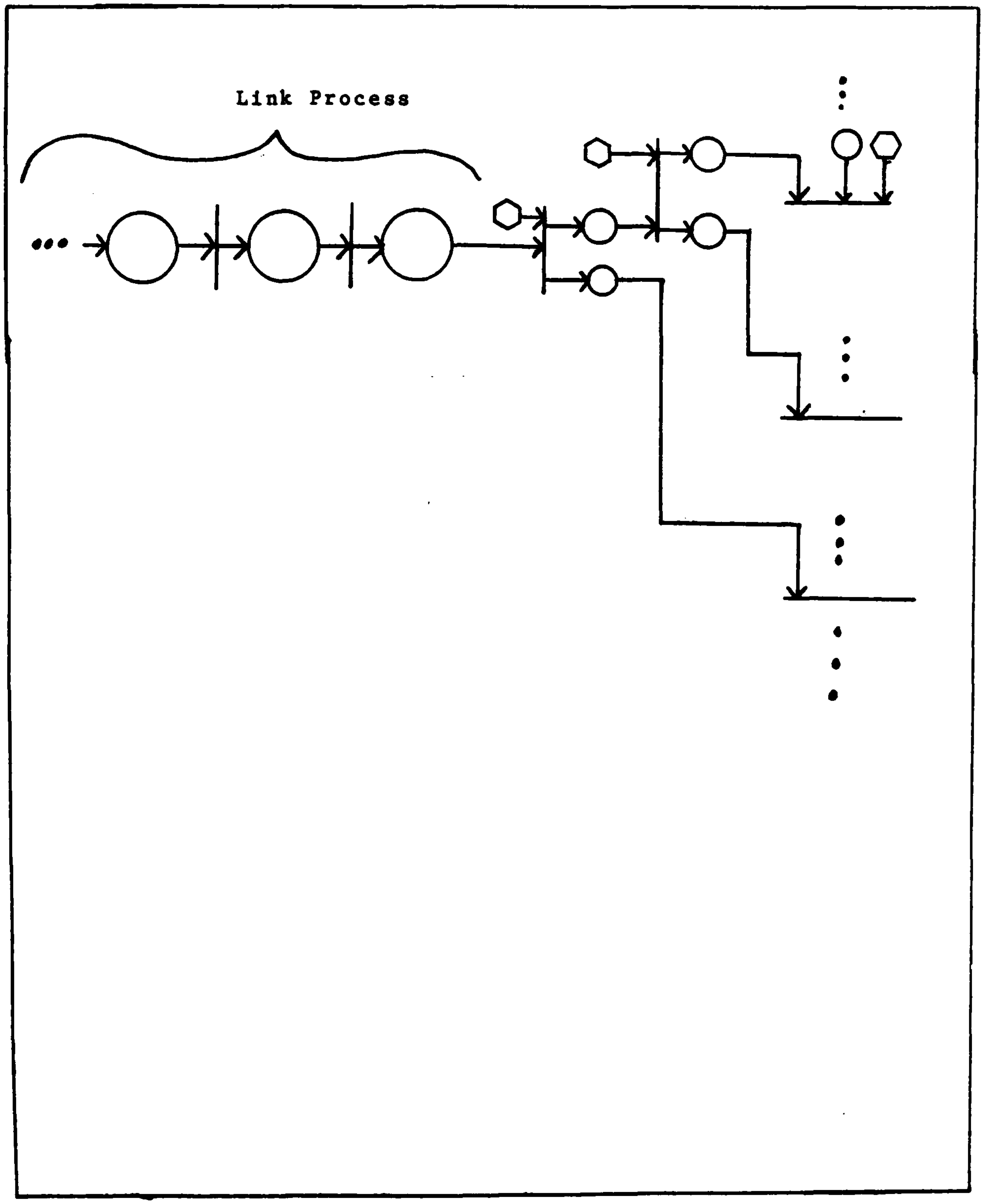
a) Single RECEIVE from a link.



b) Multiple RECEIVEs from same link.

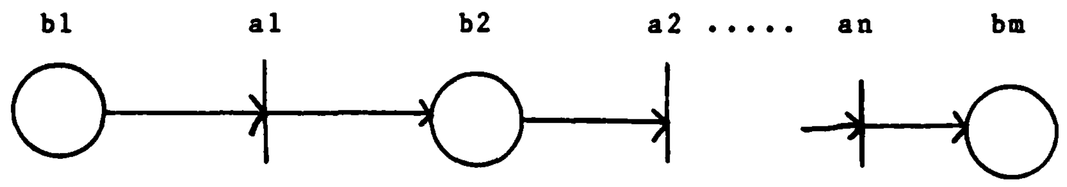
<u>Multiple RECEIVE Statements to Same Link</u>

<u>Figure 4-13</u>



Multiple RECEIVEs to same link process

Figure 4-14



Link Process Primitive.

Place b1 is the place that receives tokens from SEND statements referencing this link. Transition an is the Linkend transition. Place bm receives a token when a message is available to a RECEIVE statement referencing this link.

E-Net Construct For PPML Link Process

Figure 4-15

E-Net Evaluation

The second stage of the evaluation is the actual E-Net simulation. Since we are dealing with an underlying E-Net structure the means of simulating the action of the protocol is already pretty well defined. Using the rules associated with transition enabling and firing we can model the dynamics of the protocol. The main question to be answered in this phase was what data should be collected about the simulation to support the evaluation of the protocol.

The basic approach to the actual exercising of the net is the event driven simulation. This form of simulation maintains a chronological list of events that are scheduled to take place. Since the E-Net incorporates the concept of finite transition firing times we can use this approach. The events in the E-Net are the transition firings. When a transition becomes enabled a firing time is computed and the event associated with the firing of the transition is inserted into the event list. The eventlist is kept in ascending chronological order so that the next transition to fire is always first on the list.

When an event is removed from the list, the transition associated with the event is fired. The transition procedure for the transition is executed and the token movement specified by the transition type is accomplished. After the transition fires there will probably be new transitions made enabled. So, after a transition fires the transitions that have their input places coincident with the output places of the fired transition are evaluated to see if they are now enabled and should be put on the eventlist. That is the way the E-Net is exercised. The other consideration during this phase was the specification of information to be collected for evaluation of protocol performance.

The two main data items that can be collected from the E-Net are the states of the model as represented by the token placement as the net is exercised and information on token transmission time. The states of the system are maintained in a "token machine". The token machine is a representation of all the states the E-Net has progressed through.

The other type of data to be collected is statistical in nature. We are interested in evaluating the timing characteristics of the protocol which leads us to the desire to have some information on the transmission times of the messages. One way to do this is to keep a record of when the message was sent and when it was successfully received. All that is needed is to record the time sent and the time received in the token representing the message. These will then become two more attributes associated with all message tokens.

Summary

A procedure for describing a protocol in PPML statements has been designed. A method of converting the PPML description into an equivalent E-Net has been developed and designed and a way to exercise the E-Net has been designed. The next step is to implement the design in a programming language.

V IMPLEMENTATION

Introduction

It was decided early in the project planning stages that the final implementation would be a program written in Pascal on the VAX 11/780 Scientific Support Computer using the UNIX operating system at the Air Force Institute of Technology school of Engineering. The two reasons for choosing Pascal are that it was designed as a tool to further the concepts of structured programming and it was the only language available at the time that would allow easy implementation of dynamic data structures. The primary reason for choosing the UNIX system was that it incorporated facilities for the automatic maintenance of computer programs via the operating system command ´make´. This facility allows automatic updates to large programs.

It also became evident quite early in the project design that the underlying structure of the implementation would be lists of information. The lists would contain data input by the operator like the Message formats, Global variables, etc. Lists would also be system generated as the E-Net is exercised like the Token Machine, Active message list, Event list, etc.

Besides the list maintenace function of the implementation the problem of how to actually represent the E-Net structures had to be addressed. This dealt with designing a Pascal representation of the transition procedures and resolution procedures.

First a description will be given of how the various lists are built and manipulated followed by a discussion of how the E-Net constructs are implemented in Pascal.

List Implementation

The basic format for the data structures is a header that points to the first entry in the list followed by the acutal elements in the list. This can be seen in the pictorial representation of the lists in Appendix C. A typical example is the Message format list shown on page C-1. The header is labeled MSGLIST. This in turn points to the message header portion of the first message type. The message header has two pointers. One points to the next message header and the other points to the first field under the header. Each field then points to the next field associated with this message description.

The process of building the message format list consists of getting the necessary information from the user and creating the appropriate type of record and attaching the record to the list in the proper position. The sequence of building a typical message format is as follows. When the user selects the message format menu and the new message selection, the system prompts for the message type. When this information has been gathered a record consisting of the message type and other bookkeeping information is built and attached to the previously built message header. The system then prompts for information on each field to be associated with this message. As each field is described, a record is built to hold the data and is subsequently attached to the list of field descriptions for this message. This process repeats until the user selects the exit option from the message format menu. This process is typical of all lists that are created with the help of user input. These are the message format, global variable and process lists.

The system generated lists include the transiton, place, active message, token machine and event lists. These lists are created in response to the building and exercising of the E-Net. All of these were fairly straightforward to implement except the transition list. I will describe each of these in more detail as they impact the analysis of the protocol to a higher degree than the lists on the previous section.

The place list consists of a list of records representing the places of the E-Net. The major aspect of a place description is its marking. The marking of the place must define its status as either full or empty as well as indicate which token resides in the place if it is full. The means to model the marking is ideally suited to a pointer variable. The pointer will point to the active message that resides in it or will have the value of ´nil´, indicating that it is empty. This form of implementation facilitates detecting the full places when building the token machine. The place list can be scanned to detect the places that have tokens and can readilly provide information on which token resides there.

The active message list is built in response to the PPML 'SET' statement. When a new message is built it is placed on the active message list. A pointer to it is returned and access to this message will be through the manipulation of this pointer value. This form of implementation of the active messages allows after the fact analysis to be readilly accomplished. Information as to transmission times and errors can be collected at the end of a session and analyzed on or off line.

As a consequence of the form of the place records the token machine is easilly built. When a transiton fires, the place list is scanned for the places that have tokens residing in them. This information is then assimilated into one list representing the state of the E-Net at that time and appended to the token machine.

The event list contains information as to which transitions are enabled and when they will complete their execution phase. The list is kept ordered so that the next transition to fire is always at the head of the list. This ordering precludes having to search the entire list to find the next transiton to fire.

Transition implementation

The question as to how to implement the transition descriptions was not as straightforward to solve as the other issues during this phase of the development. It was evident that information concerning input and output places would have to kept. This information is kept both as numerical data and pointer data. The place numbers as well as pointers to the place data structures are recorded in the transition data structure. The problem came when trying to decide the form of the transition procedure.

The first form considered was a direct implementation of the transiton procedure as defined in chapter 4. This would require another data structure appended to each transition to define the transition procedure as well as a means of interpreting the transition procedure when it came time to fire the transition. Although this would be possible, it meant developing another compiler type process that was beyond the scope of this project. So, an equivalent means of representing the transition procedures that would be easier to implement was sought. The search began by looking at the form of the PPML process description and the E-Net. The first hint as to the solution to this problem came by looking at the effective results of each primitive.

The first primitive looked at was the SET construct. The effect of the transition procedures associated with this structure is to access the message format list and build an active message using the message format as a guide. With this in mind, the SET could be reduced to one transition. When the transition became enabled and fired a message representing the prescribed format as specified in the PPML statement would be built and the pointer to this newly created active message would be passed to the output place of the SET transition. So the only information necessary to accomplish the SET instruction is the message format to be used. This led to having a pointer to the message format header of the specified message in the transition procedure. When the transition fires all that has to be done is to access the message format list at the message pointed to by the transition procedure. This resulted in the SET instruction being modeled by a single transition. This process was applied to the other primitive constructs with similar results. All PPML instructions ended up being modeled by single transitions with transition procedures tailored to each PPML statement.

The information needed by the SEND transition was the place that represented the input to the link specified in the PPML statement. The only question concerning this process was could it be applied if there were multiple SEND statements within the same process referencing the same link? This turned out to be a viable solution. The primary reason this would work is that the processes are essentially sequential in nature. This means that only one SEND statement accessing a link will be active at a time. Accordingly, there will never be the situatuon where a transition tries to place a token in a place that already has a token. Due to the underlying sequential nature of the processes we can have more than one transition placing tokens in a single place.

As a result of the analysis of the SEND statement a simpler way to model multiple input transitions to a single place became evident. The elaborate structure developed in Chapter 4 to allow multiple transitions to be linked to the same transition as in the branching statements would not be required.

Another result of this analysis was the incorporation of the resolution procedure in the transition procedure of the transtion connected to the resolution place. The only type of PPML statements needing the resolution locations are the IF and UNLESS statements. The predicates associated with these statements can be kept in the transition procedure and evaluated during transition firing and the appropriate action taken as to which output place gets the token.

Output Implementation

The final phase of the implementation dealt with exercising the E-Net and presenting the results of the simulation effort. The two primary data items to be collected were the Token Machine and message transmission times. In support of the two primary items the E-Net structure as well as the global variable values and active message list are made available to the user during the execution of the E-Net. All the aforementioned items are made available through the menu displayed during the

E-Net evaluation. Once the simulation has completed the token machine and messages can be examined to determine the pertinent characteristics of the protocol.

Summary

The project was implemented on the VAX 11/780 under the UNIX operating system using Pascal. The bulk of the implementation effort dealt with the handling of linked lists. The implementation proceeded in a top down fashion as a reflection of the top down design. Modules were tested in a limited way as they were coded with the overall testing reserved until after the whole system had been coded.

VI SYSTEM TESTING

Introduction

The purpose of testing the code is to install a level of confidence in the implementation of the design. In order to accomplish this a testing procedure needs to be developed as the system is being designed. The testing procedure should cover the design issues as presented in the requirements definition as well as test the consequences of error conditions.

The testing of this implementation proceeded in two phases, modular and system testing. This chapter will describe the two procedures as applied to this project and the results of each.

Modular Testing

The concept of modular testing is an attribute of the top down design process and implementation. It is a natural extension of the top down modular design. As a module is coded it is tested in concert with existing modules. If the newly developed module contains calls to modules that have not been implememted then module "stubs" can be used to simulate the action of the called module. Stubs can return values that will typically be returned when the actual module is called or may indicate that it has been called without returning any value. The method used in this project was to have the unwritten modules indicate when they were called without returning any value. An example of how the modules were tested can be taken from the test of the highest level of the system. When the module that is represented by structure chart A-1 was coded stubs were used by the subordinate structres. Since the only function of this module is to call the other modules it could be tested by having modules 1.3 - 1.5 consist of a statement that wrote to the terminal that it had been called and then return to the calling module. A typical example of this would be that module 1.3 would write "Message format called" on the terminal.

The modular testing is implemented as the system is coded. Each module is tested to check that it interfaces correctly with the modules that call it and to the modules it calls. This type of testing gives some assurance that the modules operate correctly with respect to their immediate neighbors. In order to test the overall system an integrated form of test is required. This test comes in the form of the system testing procedure.

System Testing

System testing is the process that is used to examine the overall system to determine if the implementation accomplishes the goals set forth in the requirements definition. System testing is accomplished after the implementation and integration of all modules has been completed. The procedure used to test the system was developed during the requirements definition phase and amended during the design phases. The concept of the system test was to test the system with valid input as well as invalid inputs. In order to accomplish the system test a small protocol was fabricated. A small one was required so that the E-Net that was built could be validated against the manually built E-Net. The results of the test are shown in Appendix D. A description of the test follows.

<u>Message Input</u>

1) Test that any message format could be input.

2) Results on page D-2.

3) Comments: Of course the test could not cover every conceivable format. A representative number of examples were input and the results of one of them is shown on D-2. It was demonstrated that messages with any number of fields could be input.

<u>Delete Message</u>

1) Test that a message format could be deleted.

2) Results on page D-3.

3) Comments: This test also tested the listing of the message list when the list was empty.

<u>Delete Nonexistent Message Format</u>

1) Delete message that is not on the message list.

2) Results on page D-4.

3) Comments: It indicates that the specified message type does not exist. No fatal error.

<u>Illegal Function Number Entry</u>

1) Test an out of range response to Master Menu.

2) Results on page D-5.

3) Comments: No fatal error unless noninteger input.

<u>Illegal Response to Master PPML menu</u>

1) Test out of range input.

2) Results on page D-6.

3) Comments: No Fatal Error unless non integer is input.

<u>Illegal Response to Global Variable Menu</u>

1) Test out of range input.

2) Results on page D-7.

3) No fatal error unless non integer is input.

<u>New Global Variable Input</u>

1) Test Global Variable input function.

2) Results on page D-8.

3) Comments: As with message input all possible Global variable names could not be input. Fatal error if non integer is input as initial value.

<u>New PPML Input</u>

1) Test that PPML process descriptions could be input.

2) Results on pages D-9 and D-10.

3) Comments: The test shows that more than one process can be input.

<u>Change a PPML statement</u>

1) Test the Change PPML function.

2) Results on page D-11.

3) Comments: Works as specified.

<u>Delete PPML</u>

1) Test Delete PPML function.

2) Results on page D-12.

3) Comments: Works as specified.

<u>Insert PPML</u>

1) Test Insert PPML function.

2) Results on page D-13.

3) Comments: Works as specified.

<u>Nonexistent statement number input</u>

1) Test invalid statement number input in PPML input processing.

2) Results on page D-14.

3) Comments: Returns indication that statement could not be found.

The foregoing tests tested the message format input phase and PPML statement input phase of the system. One characteristic that applies to all data input requests in this phase and all other phases as well is that if the system expects an integer to be input by the user and a noninteger is detected then a fatal error will occur and the program will abort to the operating system. The following is the test procedure used to test the E-Net building phase of the system. The processes shown on pages D-9 and D-10 were the process descriptions used.

<u>Reference to Nonexistent message type</u>

1) Test detection of nonexistent message reference in PPML.

2) Results on page D-15.

3) Comments: No fatal error. The system simply reports that it could not find the message type specified in the

PPML statement. This applies to both the SET and UNLESS statements. The test was conducted by deleting the specification of message type ACK1 from the message list.

E-Net Building

1) Test that E-Net building progresses when no errors are present in process description.

2) Results on page D-16.

3) Comments: Statement numbers are shown as each statement is processed.

Invalid PPML statement detection

1) Test consequences of processing when an invalid PPML statement type is detected.

2) Results on page D-17.

3) Comments: Error message displayed. A fatal error was detected during the test. If the invalid statement precedes one that has a label then when it becomes time to process any jumps to the transition, there is no input place to reference.

Reference to Nonexistent link name

1) Test consequences of referencing a link name that is not on the link list.

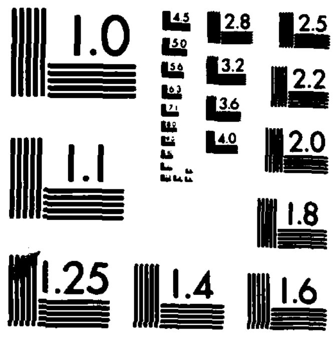
2) Results on page D-18.

3) Comments: Non fatal error. Error is detected and reported.

The last phase of the testing process dealt with the E-Net execution phase.

AD-A138 160 AN AUTOMATED COMPUTER COMMUNICATION NETWORK PROTOCOL VERIFICATION SYSTEM(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING 2/3

UNCLASSIFIED K R MARTIN DEC 83 AFIT/GCS/EE/83D-12 F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

Automatic execution

1) Test Automatic net exercising.

2) Results on page D-19.

3) Comments: The type of transition being executed is reported. Fatal errors would occur if the errors reported during the E-Net building phase were not corrected.

Active Message Listing

1) Test the Active message listing function of the execution menu.

2) Results on page D-20.

3) Comments: Works as specified.

Event List listing

1) Test the Evevt List listing function of the execution menu.

2) Results on page D-21.

3) Comments: Works as specified.

Show E-Net

1) Test Show E-Net function of execution menu.

2) Results on page D-22.

3) Comments: Link process transitions are shown at the head of the listing with numbers greater than 100.

Manual Execution

1) Test the Manual execution of the E-Net.

2) Results on page D-23.

3) Comments: By selecting option 5 from the manual execution menu the event at the head of the eventlist is processed.

Show Token Machine

1) Test Show Token Machine function of execution menu.

2) Results on page D-24.

3) Comments: Works as specified.

Show Token Placement

1) Test Show token placement function of execution menu.

2) Results on page D-25.

3) Comments: Works as specified.

Summary

There is no way to test all possible combinations of inputs to the system. A representative sample of inputs including correct and incorrect was tested. The results of the tests are shown in Appendix D. The overall result was that the system does operate according to specifications except where noted otherwise. The analysis of the outputs was not implemented so it could not be tested.

VII RESULTS AND RECOMMENDATIONS

Introduction

This chapter will present a summary analysis of how well the system implemented meets the objectives of the project and some recommendations for further work in the area. The user interface features will be discusssed followed by a summary of the functional characteristics of the system.

Results

A procedure for specifying a protocol has been presented and developed. This method uses a modified subset of the Program Process Modeling Language(PPML). Each communicating entity is represented by a "Process" consisting of a list of PPML statements. The processes communicate via channels called "Links". Messages are sent from one process to another via a link. All links are "one way" only. This system allows any number of communication processes to be modeled and analyzed.

The second major result of the project is a method has been developed to convert a PPML description into an equivalent E-Net. The E-Net can then be used to simulate the action of the protocol. The simulation is carried out by exercising the E-Net using the execution rules of transition firings. As the E-Net is being executed, information useful for subsequent analysis is collected.

This information includes a Token Machine and transmission times of the messages. The token machine is a representation of the states through which the protocol system has progressed. The information concerning transmission times can be analyzed to examine the performance of the system in terms of time delays.

Feature Assessment

The features of user friendliness and robustness are the two main characteristics of user interface that were set as goals for the system. In analyzing the user friendliness several issues become apparent. The user is not required to know in detail the inner workings of the program in order to accomplish the validation effort. He must know how to express the protocol in PPML statements and the characteristics of the communication media.

The analysis of the robustness of the system results in several observations. Syntactical errors are detected by the system and reported to the user. This feature allows the user to edit the offending statements before execution of the E-Net results in a fatal crash. There is one error that will cause a fatal crash. If an integer is expected during input and a noninteger is received the system will abort and all will be lost.

Functional Assessment

The main requirement of the system is that it detect deadlock conditions in a protocol. This system will indicate that a particular kind of deadlock has occurred. The type of deadlock referred to is caused by having the transmission of messages blocked due to transmitter buffers being full. This condition becomes obvious when analyzing the token machine because of the abnormal termination of the processes. Other conditions are not as easily recognized. The token machine of a E-Net becomes very large very rapidly because of the fact that each token is different and in general, a new state is generated by each transition firing.

Another characteristic of the system is that any topological arrangement of communication processes can be modeled. There can be an arbitrary number of processes communicating over any type of media. The system can model satellite protocols, broadcast protocols, as well as conventional protocols using landlines or microwaves.

System Resources

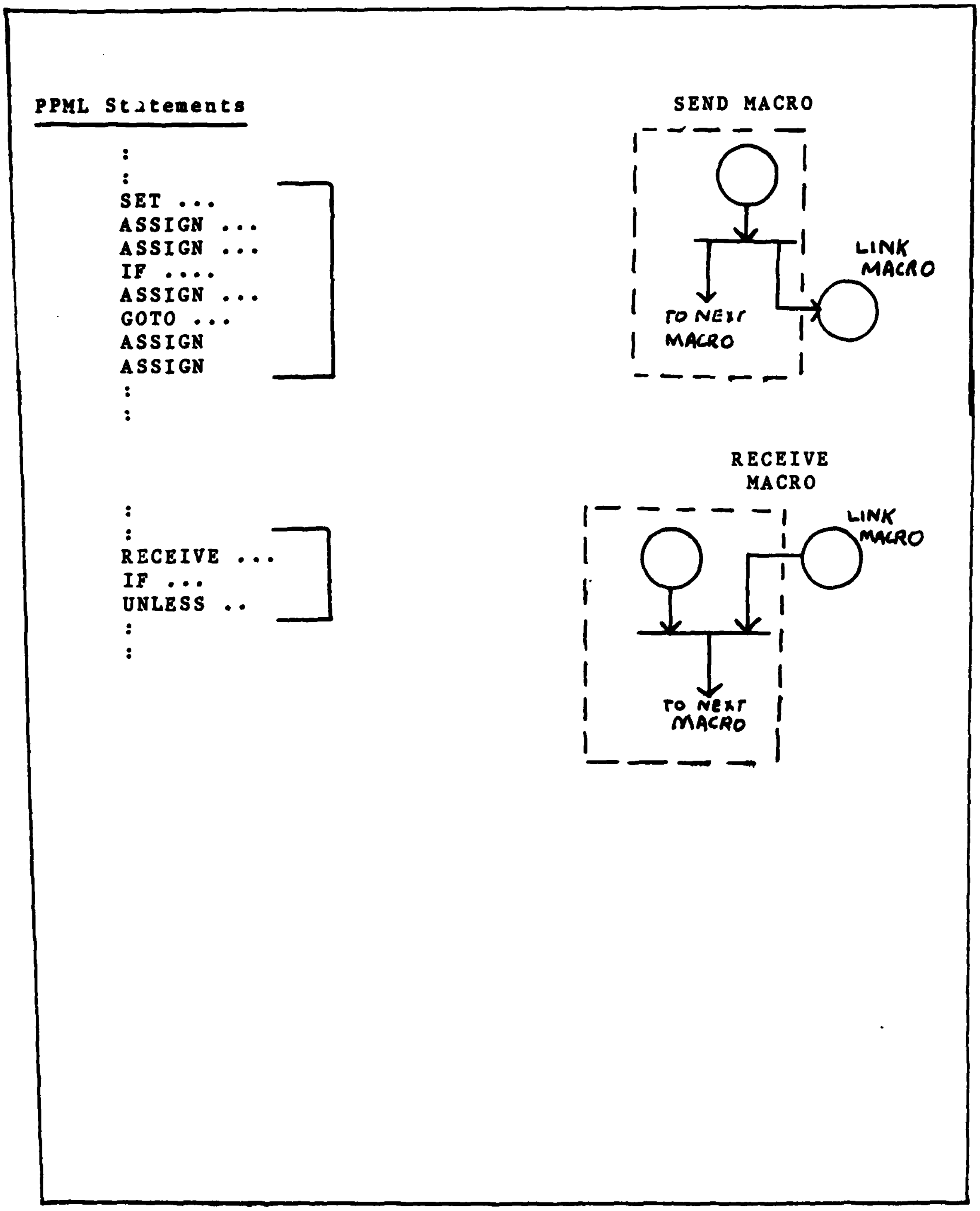
The operation of the validation system requires the VAX 11/780 with the UNIX operating system. An analysis of the operation of the system on several protocols reveals that it takes an average of 1.0 seconds of cpu time to process one hundred events in the auto execution mode.

Recommendations

The first feature that could be added is an automated token machine analyzer. The token machine has to be studied manually as it stands and this is a tedious process. It could become prohibitive due to the size of the token machine. One way to do this would be to incorporate the concept of macro E-Net constructs. This would entail grouping E-Net primitives from several PPML instructions into one macro construct. The macro would be represented as a place in the macro net. When any of the places included in the macro construct has a token then the macro would be considered to have a token. Three types of macros are all that are required. A macro for the SEND construct, a macro for the RECEIVE construct and a macro for the LINK process are the required macro constructs. A depiction of how they might be modeled is shown in Figure 7-1. The SEND macro has a place and one transition. If the macro can be exited from within then more external transitions are required. The transitions are used to transfer tokens from one macro to the next in the process. The RECEIVE macro has one place and one transition. The transition has the link as one input place and the RECEIVE macro as the other. If the macro can be exited from within then there would be more than one output transition from the macro. The link macro consists of one place. Using macro E-Net constructs for analysis shows promise and is left for further work.

Another desirable feature would be to have the capability to allow the user to specify the starting PPML statements. This would allow the simulation to more easilly model the performance of the protocol from various initial conditions.

In order to take advantage of the transmission time



PPML Statements

```
:
:
SET ...
ASSIGN ...
ASSIGN ...
IF ....
ASSIGN ...
GOTO ...
ASSIGN
ASSIGN
:
:


:
:
RECEIVE ...
IF ...
UNLESS ..
:
:
```

E-Net Macros

Figure 7-1

information collected during E-Net execution a means of analyzing transmission times should be developed. Perhaps by generating a file that could be used by one of the standard statistical packages this objective could be realized.

Although the arithmetic and relational operations incorporated are sufficient to model protocols a more comprehensive set of operators could be included to more readilly model the software of the processes. Along with this the types of data allowed in fields and globals could be expanded to include character strings and real variable types.

Summary

This project presents a protocol validation system that uses the PPML as the protocol specification language and uses the E-Net to model and simulate the protocol. Information on global state generation and message transmission times is collected during the simulation. More work is required in the information analysis portion of the system to fully utilyze its capabilities.

BIBLIOGRAPHY

1. Birke, D.M., "State Transition Programming Techniques and Their Use in Producing Teleprocessing Service Device Control Programs," IEEE Transactions on Communications, Com-20, No. 3:568-570, June 1972.

2. Bochman, G. V., "Logical Verification and Implementation of Protocols," Proceedings of the Fourth Data Communications Symposium:8.5-8.20, 1975.

3. Bochman, G. and Sunshine, C.A., "Formal Methods in Communication Protocol Design," IEEE Transactions on Communications, COM-28, No 4:624-631, April 1980.

4. Danthine, A., "Protocol Representation With Finite State Models," IEEE Transactions on Communications, COM-28, No. 24: 632-643, April 1980.

5. Harangozo, J., "An Approach to Describing a Link Level Protocol with a Formal Language," Proceedings of the Fifth Data Communications Symposium:4.37-4.49, September 1977.

6. Merlin, P.M., "Specification and Validation of Protocols," IEEE Transactions on Communications, COM-27, No. 11:1671-1680, November 1979.

7. Merlin, P.M. and Farber, D.J., "Recoverability of Communication Protocols-Implications of a Theoretical Study," IEEE Transactions on Communications, COM-24 1036-1043, September 1976.

8. Nutt, Gary J., "The Formulation and Application of Evaluation Nets." Ph. D. Thesis. Computer Science Group, University of Washington, Seattle, Washington, July 1972.

9. Peters, L.J., Software Design: Methods and Techniques New York: Yourdon Press, Inc., 1981.

10. Peterson, J.L., "Modeling of Parallel Systems," Ph.D. dissertation, Department of Electrical Engineering, Stanford University, Stanford, California, February 1974.

11. Peterson, James L., Petri Net Theory and the Modeling of Systems." Englewood Cliffs: Prentice Hall, Inc., 1981.

12. Riddle, W. E., "The Modeling and Analysis of Supervisory Systems," Ph. D. Thesis. Computer Science Department, Stanford University, Stanford, California, March 1972.

13. Stenning,V.N., "A Data Transfer Protocol," Computer Networks,Vol 1:99-110,September 1976.

14. Sunshine, C.A., Communication Protocol Modeling, Dedham: Artech House, Inc., 1981.

15. Symons, F.J.W., "Introduction to Numerical Petri Nets, a Graphical Model of Concurrent Processing Systems," Australian Telecommunication Research, Vol 14, No.1 28-32,1980.

16. Symons, F.J.W., "The Verification of Communication Protocols Using Numerical Petri Nets," Australian Telecommunication Research, Vol 14, No. 1:34-38,1980.

Appendix A

System Menus

```
          ***  MASTER MENU  ***

      1 - ENTER MSG FORMAT MENU

      2 - ENTER PPML FORMAT MENU

      3 - ENTER EVALUATE PROTOCOL MENU

Enter the number corresponding to the desired function.
```

```
              *** MASTER MESSAGE FORMAT MENU ***

               1 - Enter New Message Type.

               2 - Change a Message Format.

               3 - Delete a Message.

               4 - List Message Types.

               5 - Go to Master Menu.

Enter the number corresponding to the desired function.
```

```
          *** MASTER PPML FORMAT MENU ***

            1 - Enter Global Variable

            2 - New PPML STATEMENTs

            3 - Change PPML Statement

            4 - Insert PPML statement

            5 - Delete PPML Statement

            6 - List PPML Statements

            7 - Input Number of Communication Links

            8 - Go To Master Menu

       * Enter Desired function number *
```

```
*** MASTER GLOBAL VARIABLE MENU ***

      1 - ADD NEW VARIABLE

      2 - DELETE A VARIABLE

      3 - LIST VARIABLES

      4 - GO TO MASTER PPML MENU

   ENTER DESIRED FUNCTION NUMBER.
```

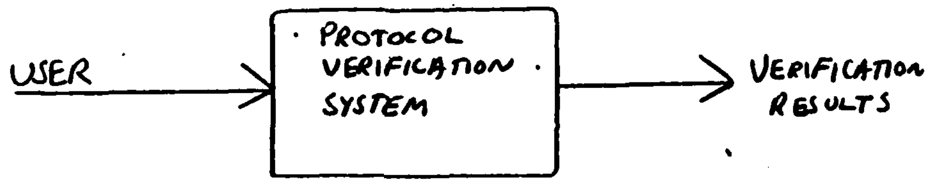
```
** EXERCISE NET MENU **

 1 - AUTO EXECUTION

 2 - MANUAL EXECUTION

 3 - EXIT

 Enter Option Number.
```

```
** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE

  2 - SHOW EVENTLIS

  3 - SHOW ACTIVE MESSAGE LIST

  4 - SHOW TOKEN PLACEMENT

  5 - CONTINUE EXECUTION

  6 - SHOW GLOBAL VAR LIST

  7 - SHOW E-NET STRUCTURE

 10 - EXIT

  Enter Option Number
```

Appendix B

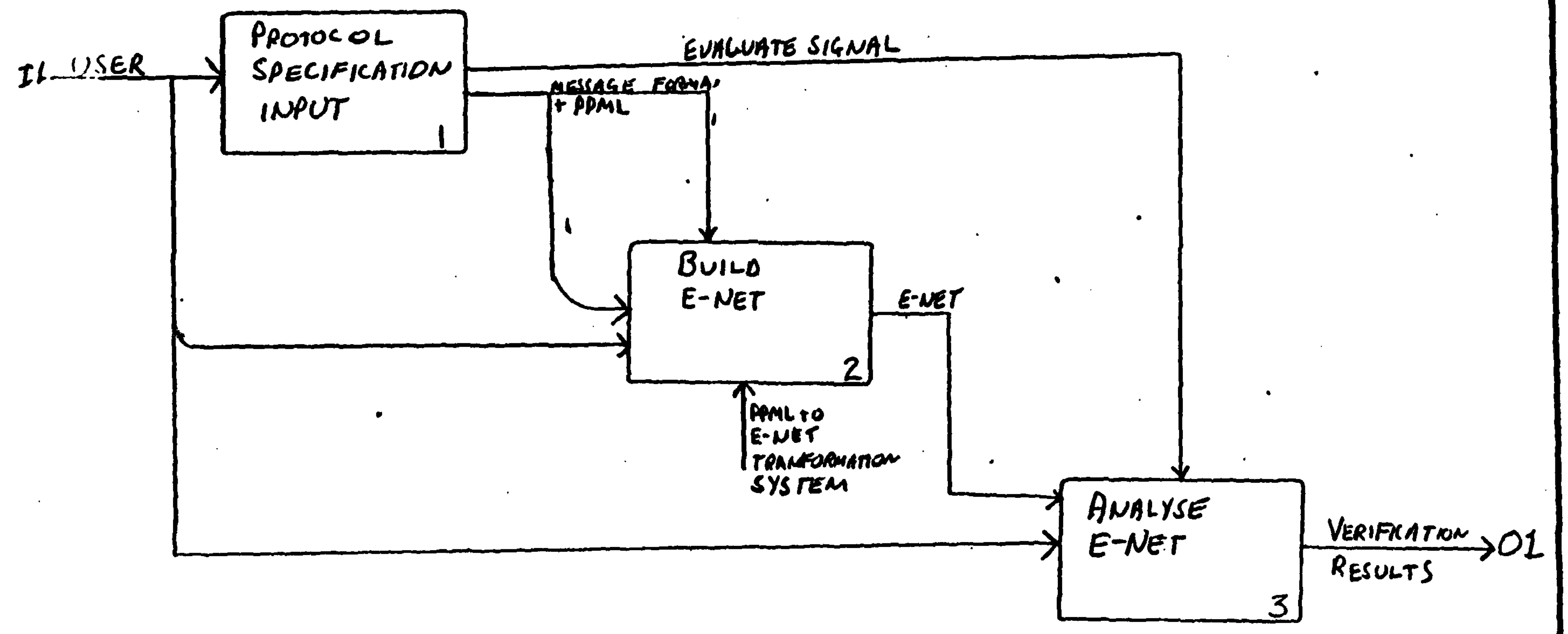
SADT Diagrams

AUTHOR:	DATE: 7 Jun 83	READER				
PROJECT: Automated Protocol System	REV:	DATE				



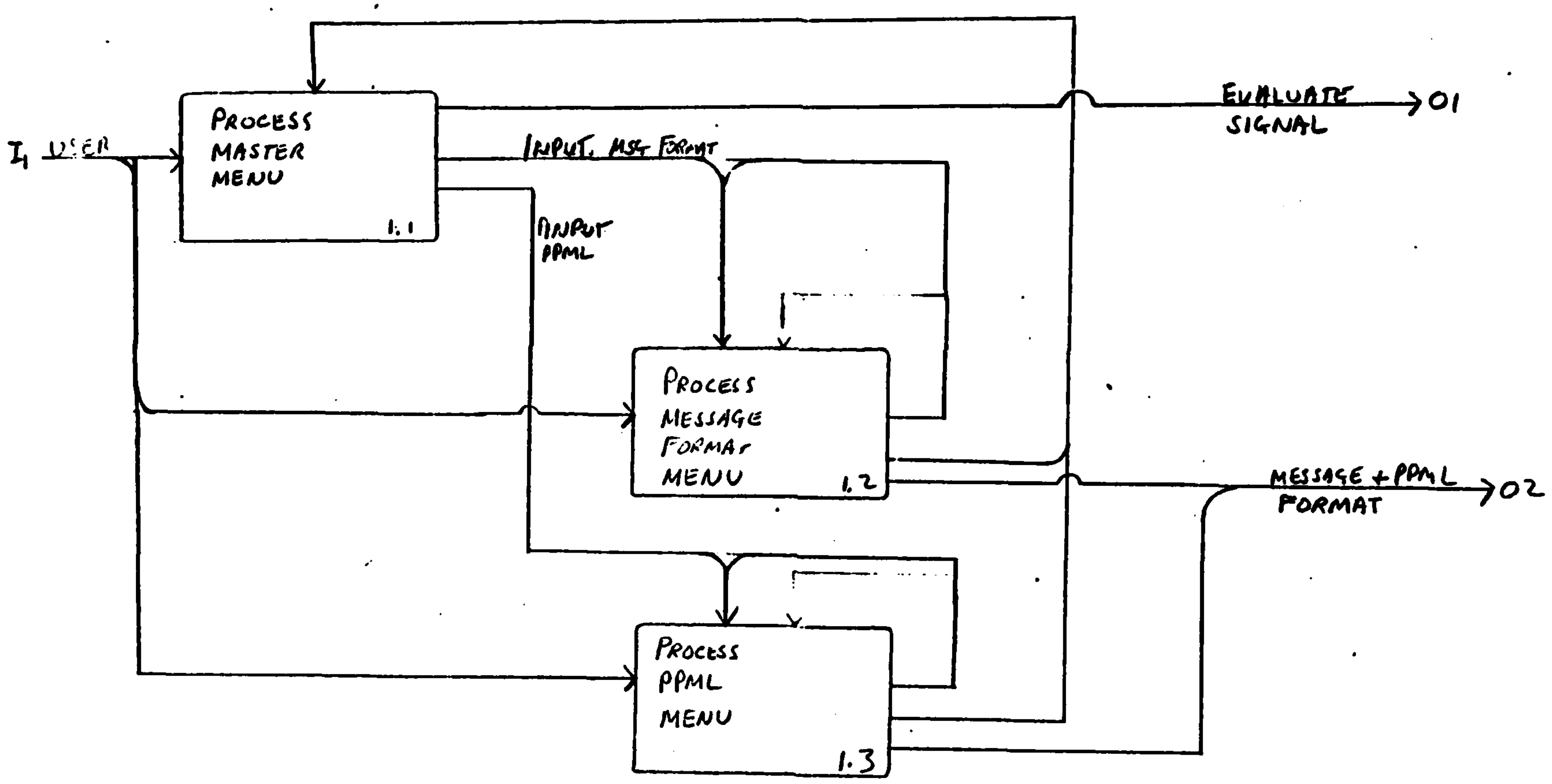
NODE: Φ	TITLE: Automated Protocol Verification System	NUMBER:

AUTHOR:	DATE:	READER				
PROJECT: Automated Protocol System	REV:	DATE				



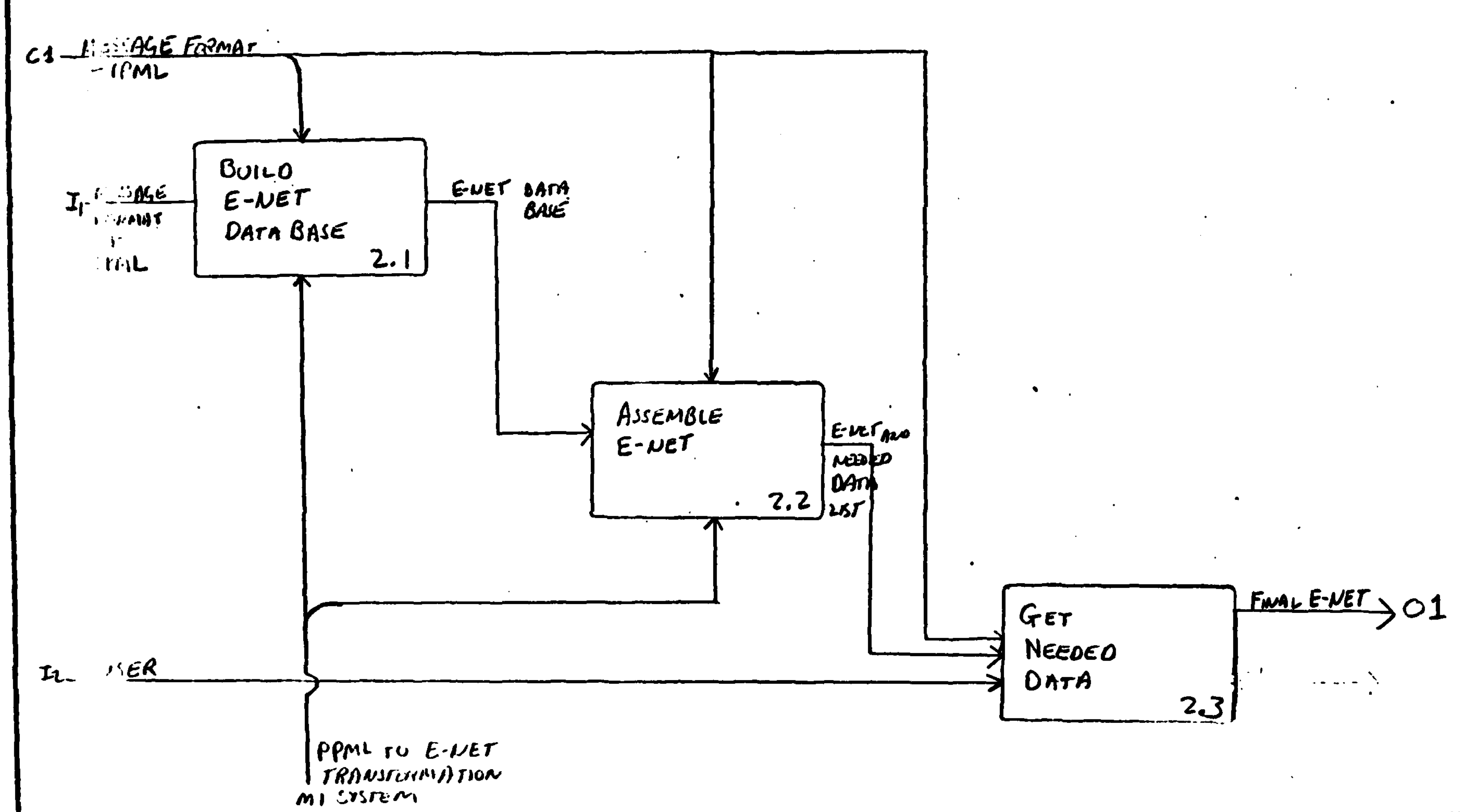
NODE: Φ	TITLE: Protocol Verification System	NUMBER:

AUTHOR:	DATE:	READER	
PROJECT:	REV:	DATE	



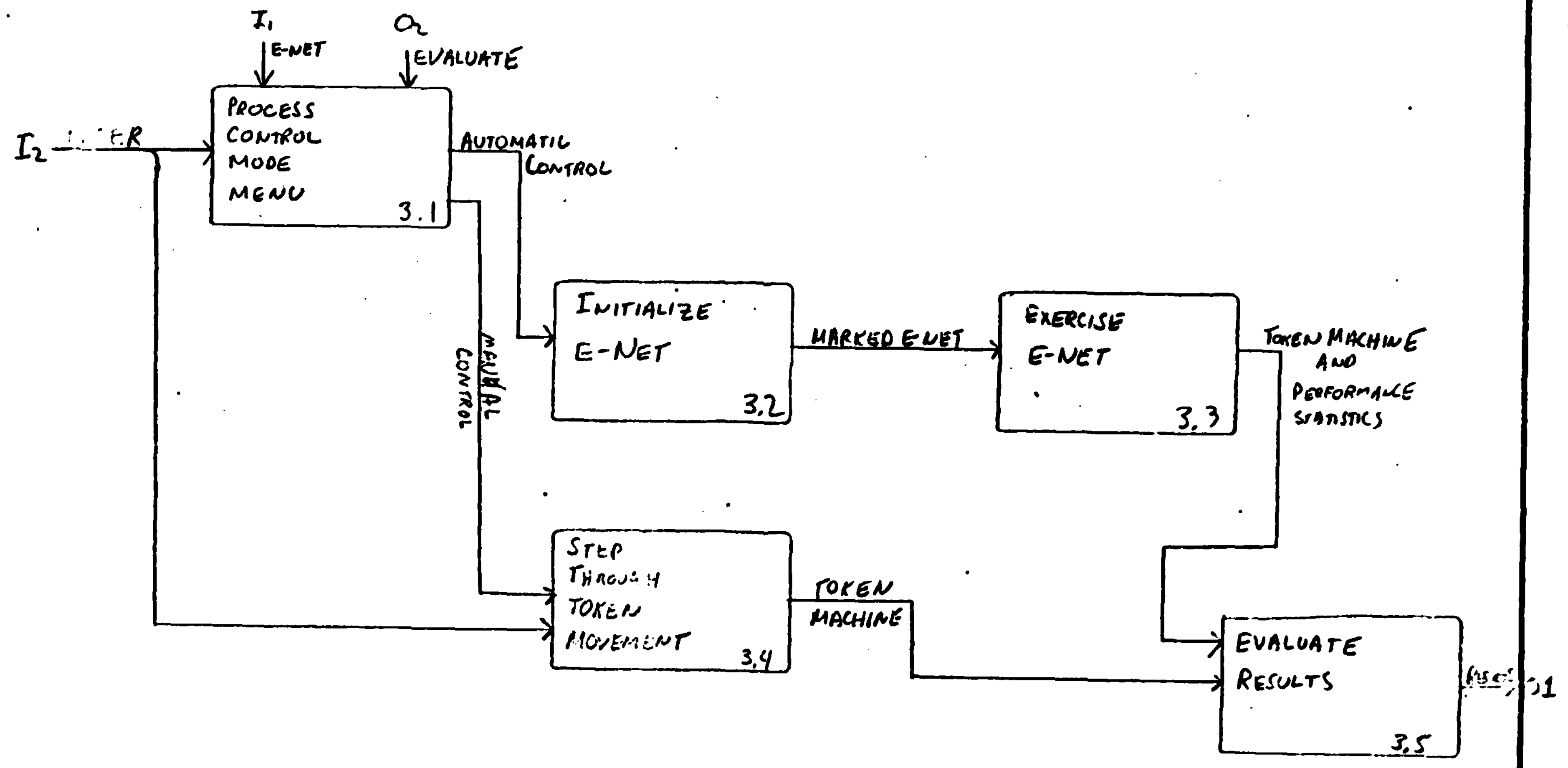
NODE: A-1	TITLE: PROTOCOL SPECIFICATION INPUT	NUMBER:

AUTHOR:	DATE:	READER				
PROJECT:	REV:	DATE				



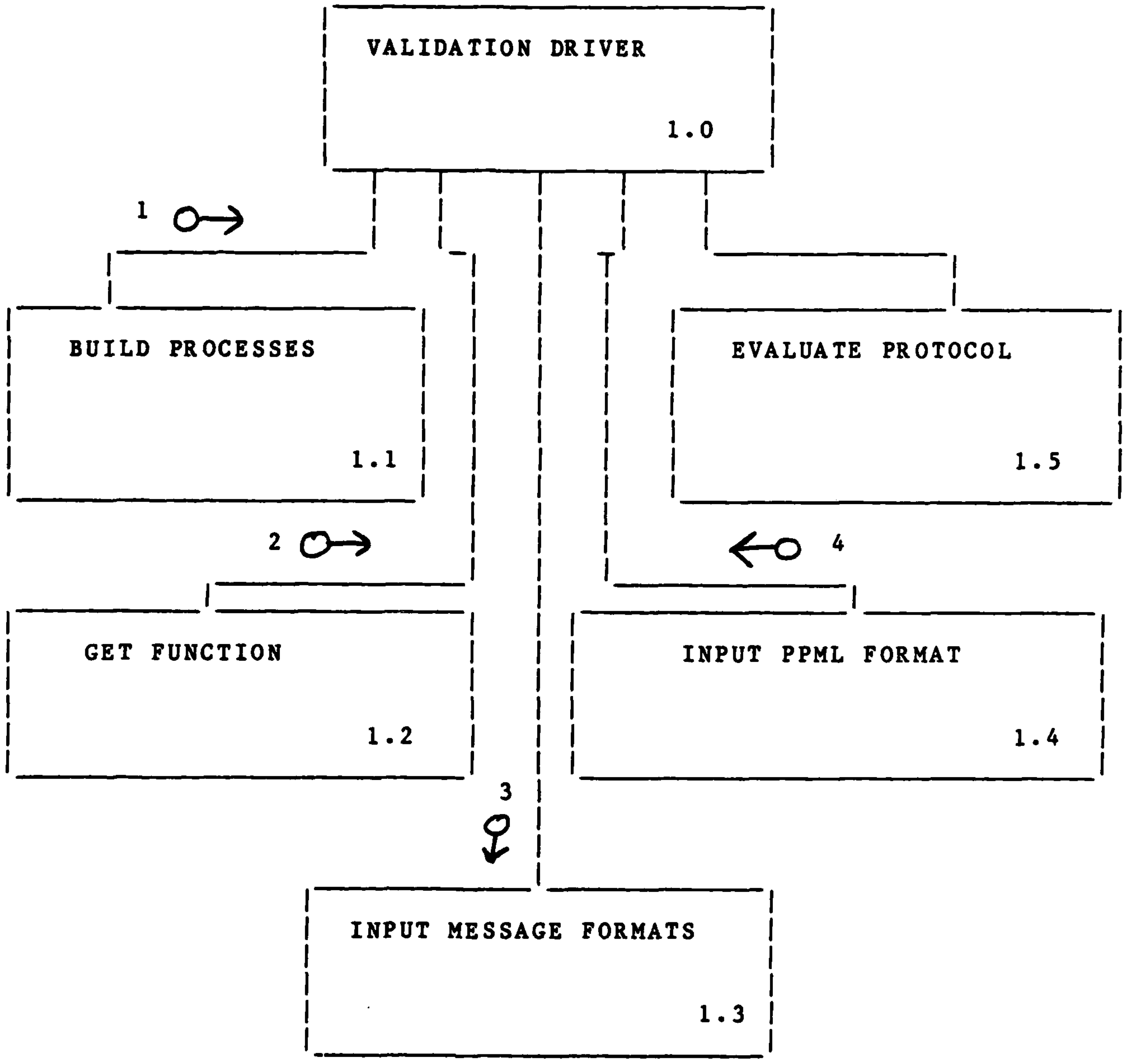
NODE: 1-2	TITLE: BUILD E-NET	NUMBER:

AUTHOR:	DATE:	READER				
PROJECT:	REV:	DATE				



NODE: A	TITLE: ANALYZE E-NET	NUMBER:

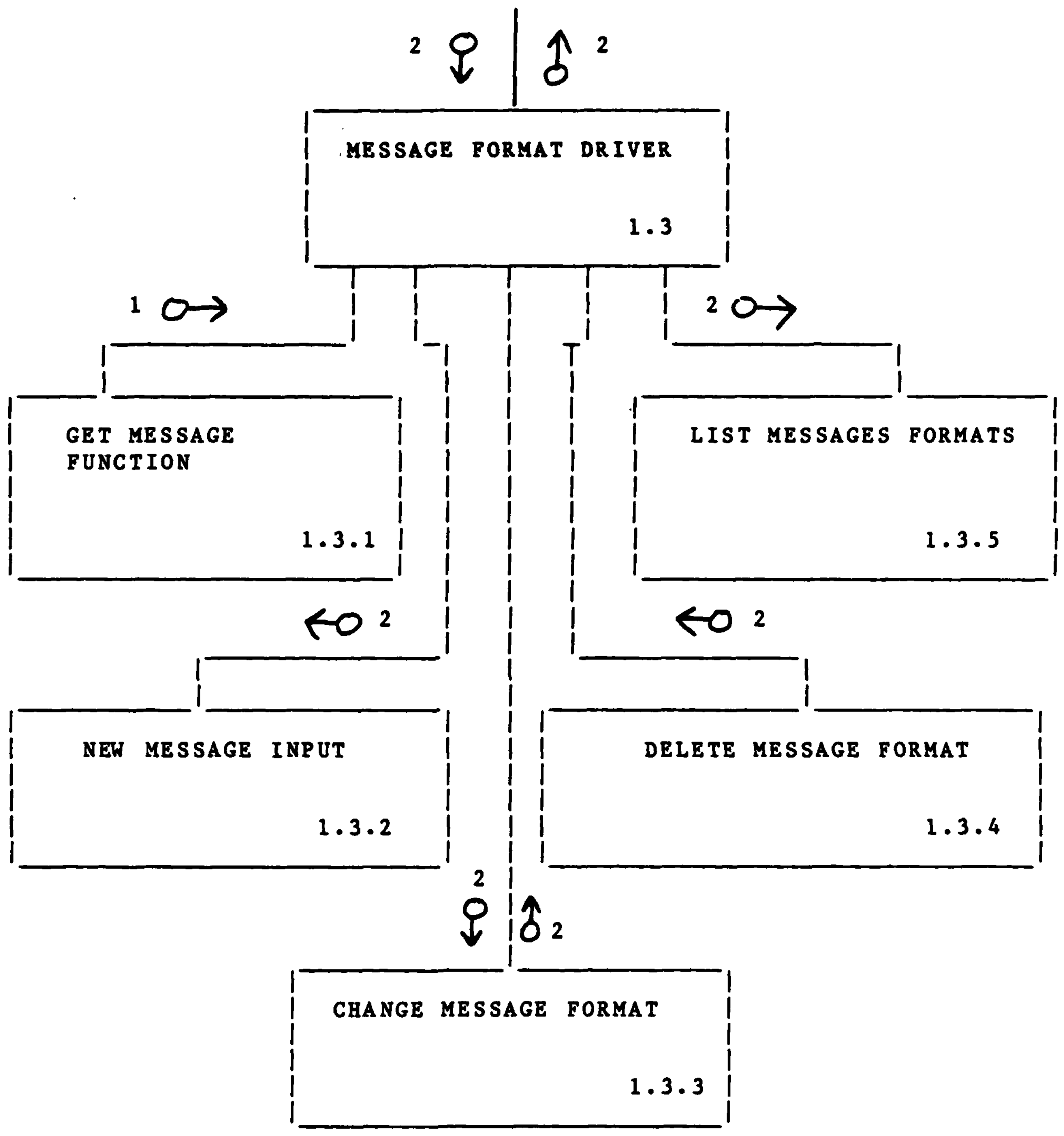
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. NUMBER_OF_PROCESSES
2. FUNCTION
3. MESSAGE_FORMAT_LIST
4. PPML_LIST

NODE:	TITLE:	NUMBER:
A-1	VALIDATION DRIVER	1

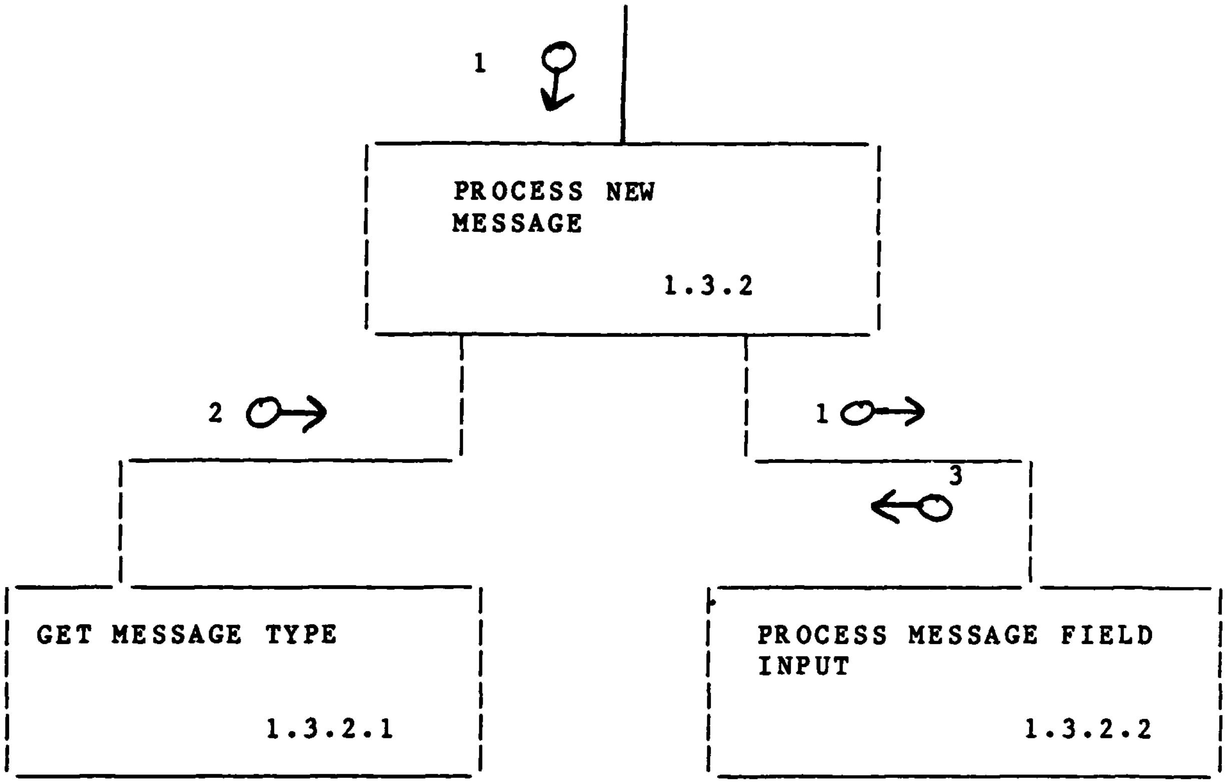
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. MESSAGE_FUNCTION
2. MESSAGED_LIST

NODE:	TITLE:	NUMBER:
1.3	MESSAGE FORMAT INPUT	2

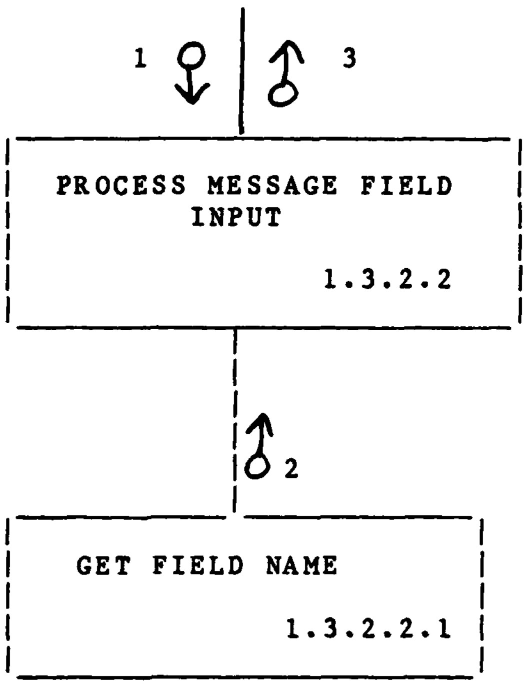
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. MESSAGE_LIST
2. MESSAGE_TYPE
3. COMPLETE_MESSAGE_FORMAT

NODE: 1.3.2	TITLE: PROCESS NEW MESSAGE	NUMBER:

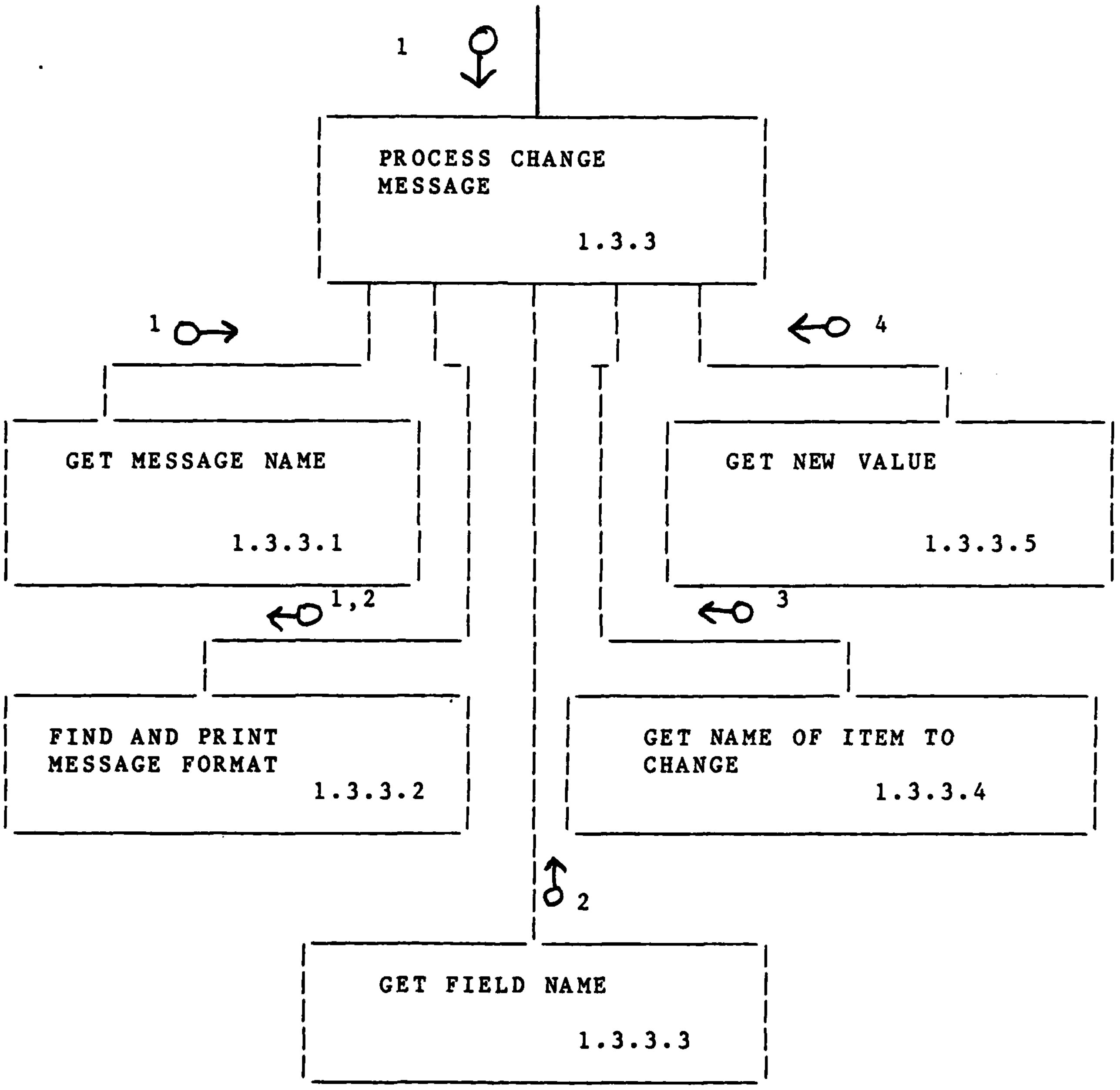
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. MSG_LIST
2. FIELD_NAME
3. UPDATED_MESSAGE_FORMAT

NODE: 1.3.2.2	TITLE: PROCESS MESSAGE FIELD INPUT	NUMBER:

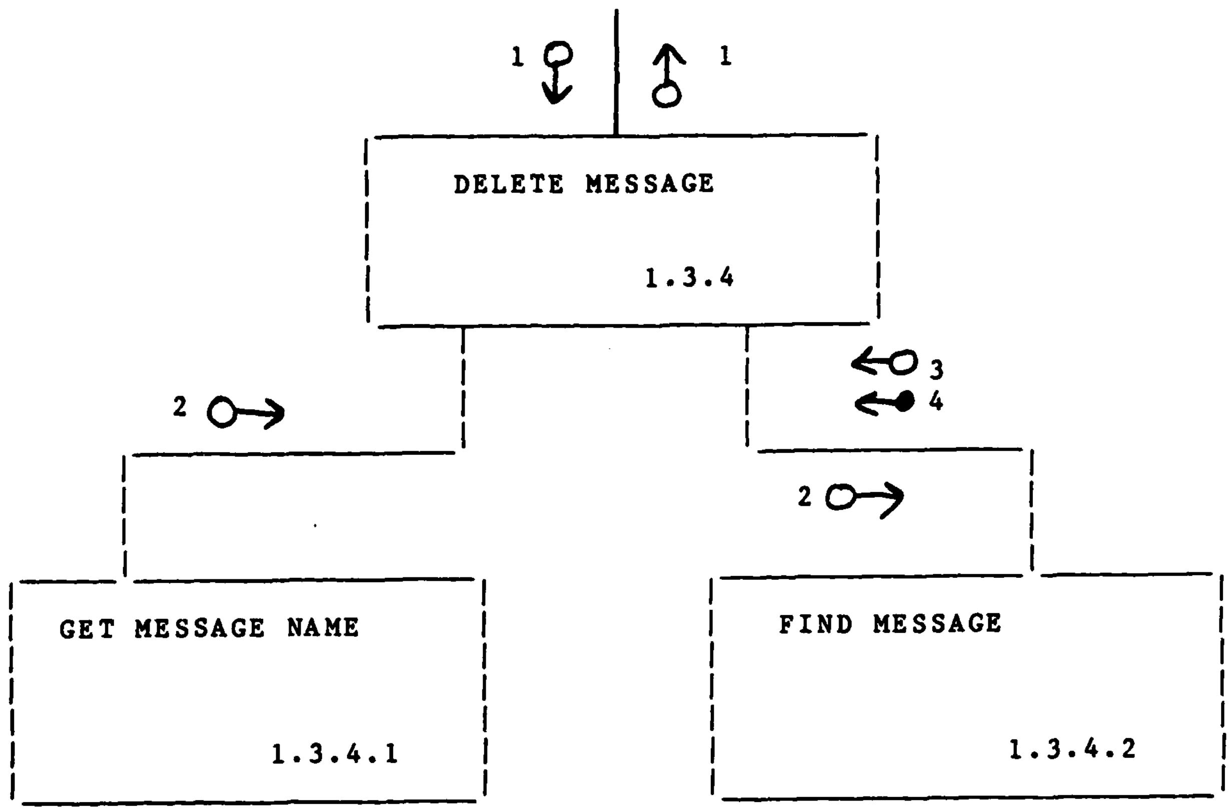
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. MESSAGE_TYPE
2. FIELD_NAME
3. ITEM_NAME
4. NEW_ITEM_NAME

NODE:	TITLE:	NUMBER:
1.3.3	PROCESS CHANGE MESSAGE	

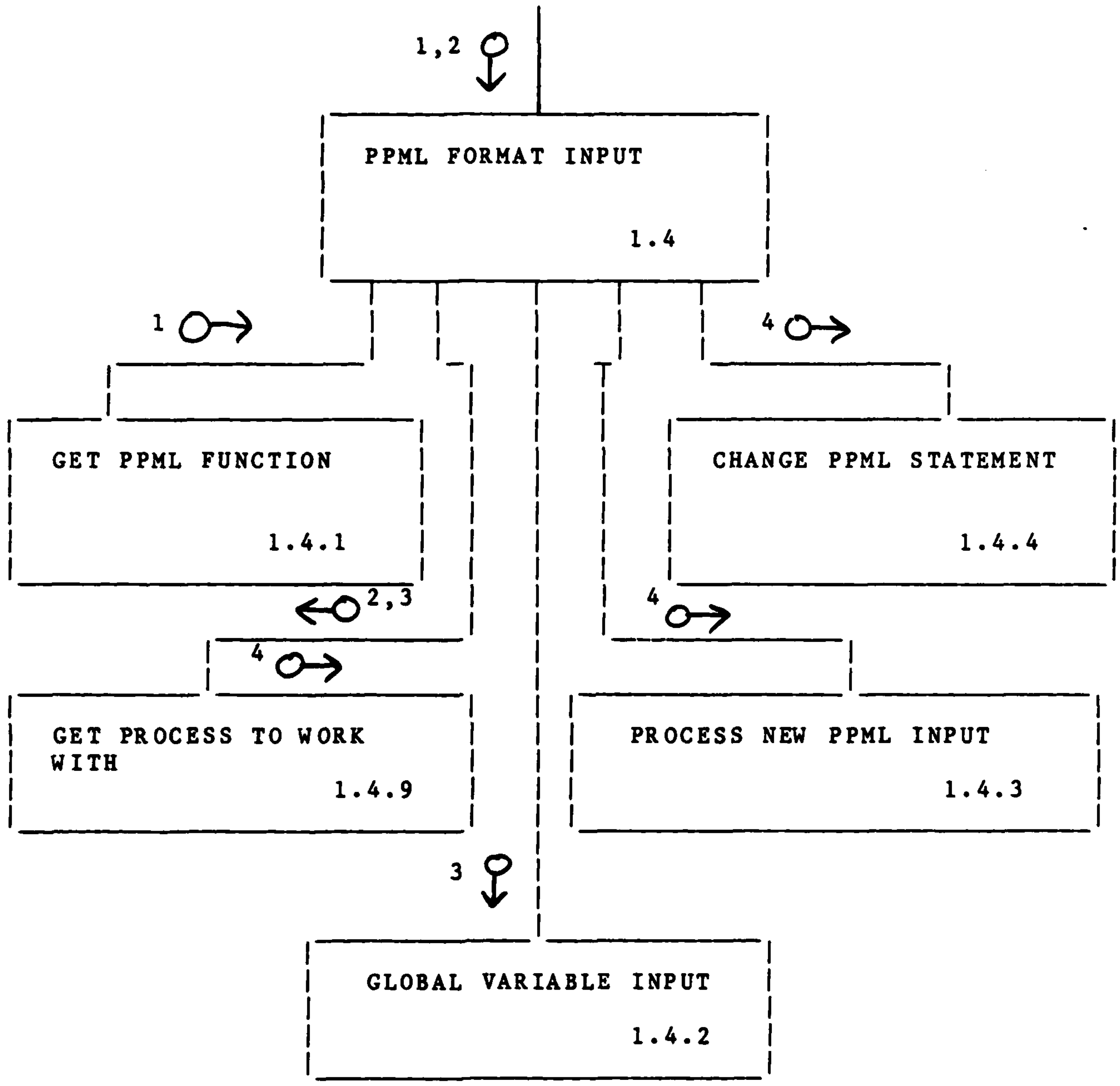
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. MESSAGE_LIST
2. MESSAGE_NAME
3. MESSAGE_POINTER
4. FOUND_FLAG

NODE: 1.3.4	TITLE: DELETE MESSAGE	NUMBER:

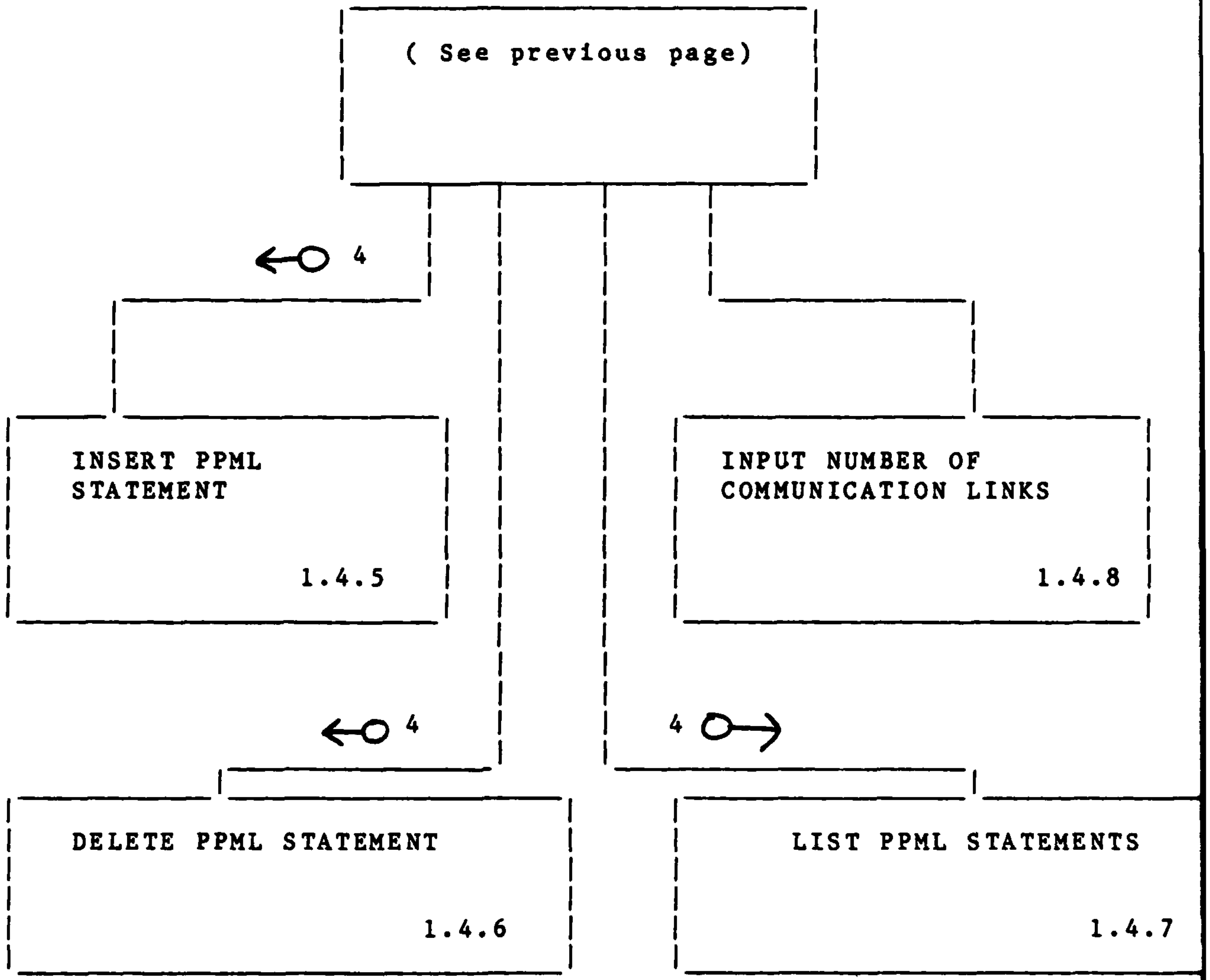
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PPML_FUNCTION
2. NUMBER_OF_PROCESSES
3. PROCESS_LIST
4. PROCESS_PTR
5. GLOBAL_LIST

NODE: 1.4	TITLE: PPML FORMAT INPUT (pg. 1)	NUMBER:

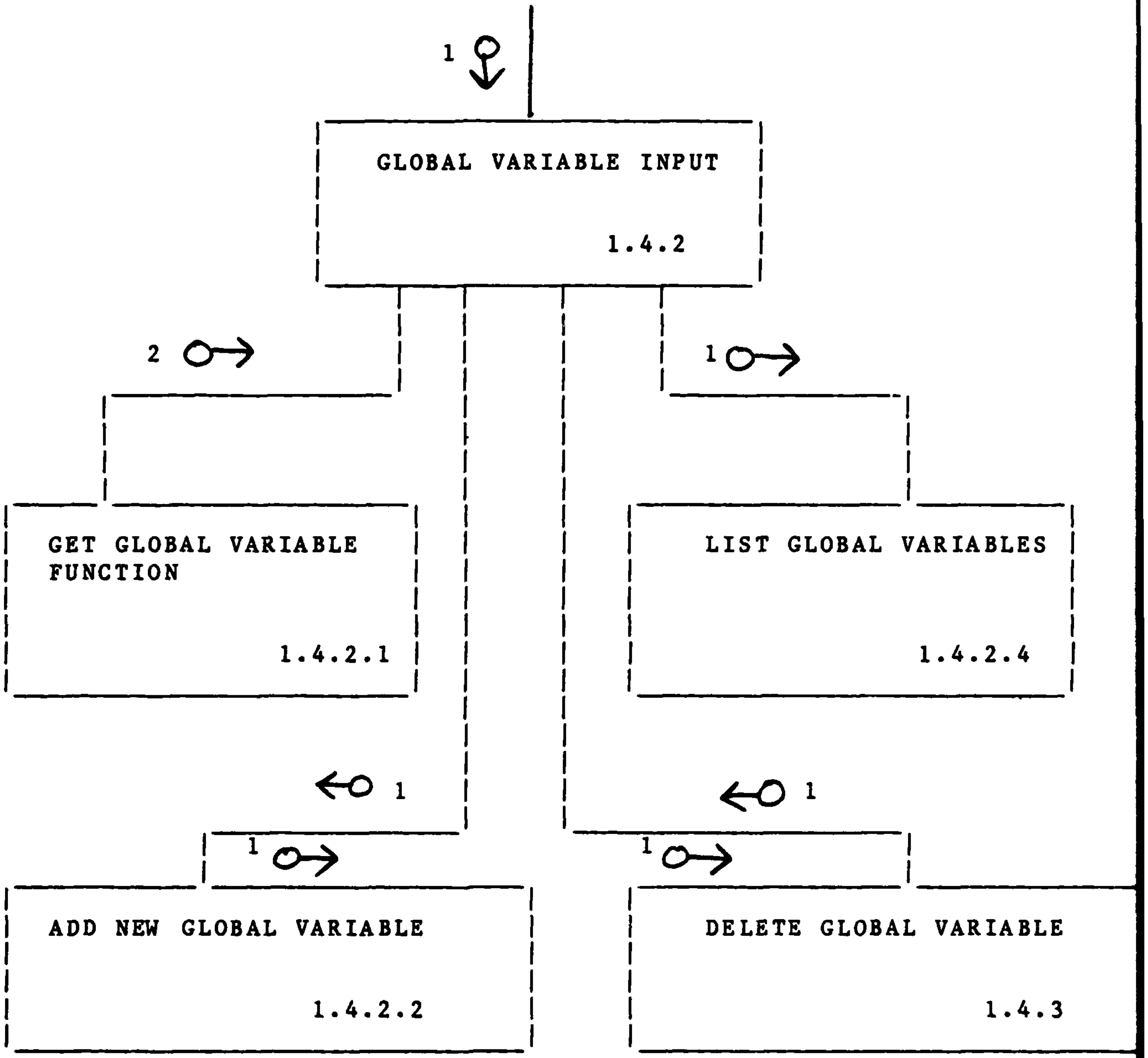
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



4. PROCESS

NODE: 1.4	TITLE: PPML FORMAT INPUT (pg. 2)	NUMBER:

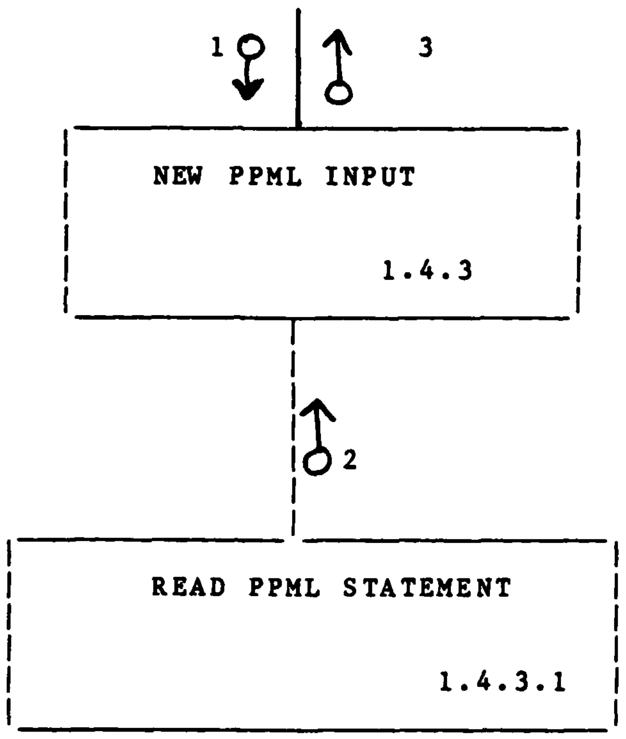
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. GLOBAL_VARIABLE_LIST
2. FUNCTION

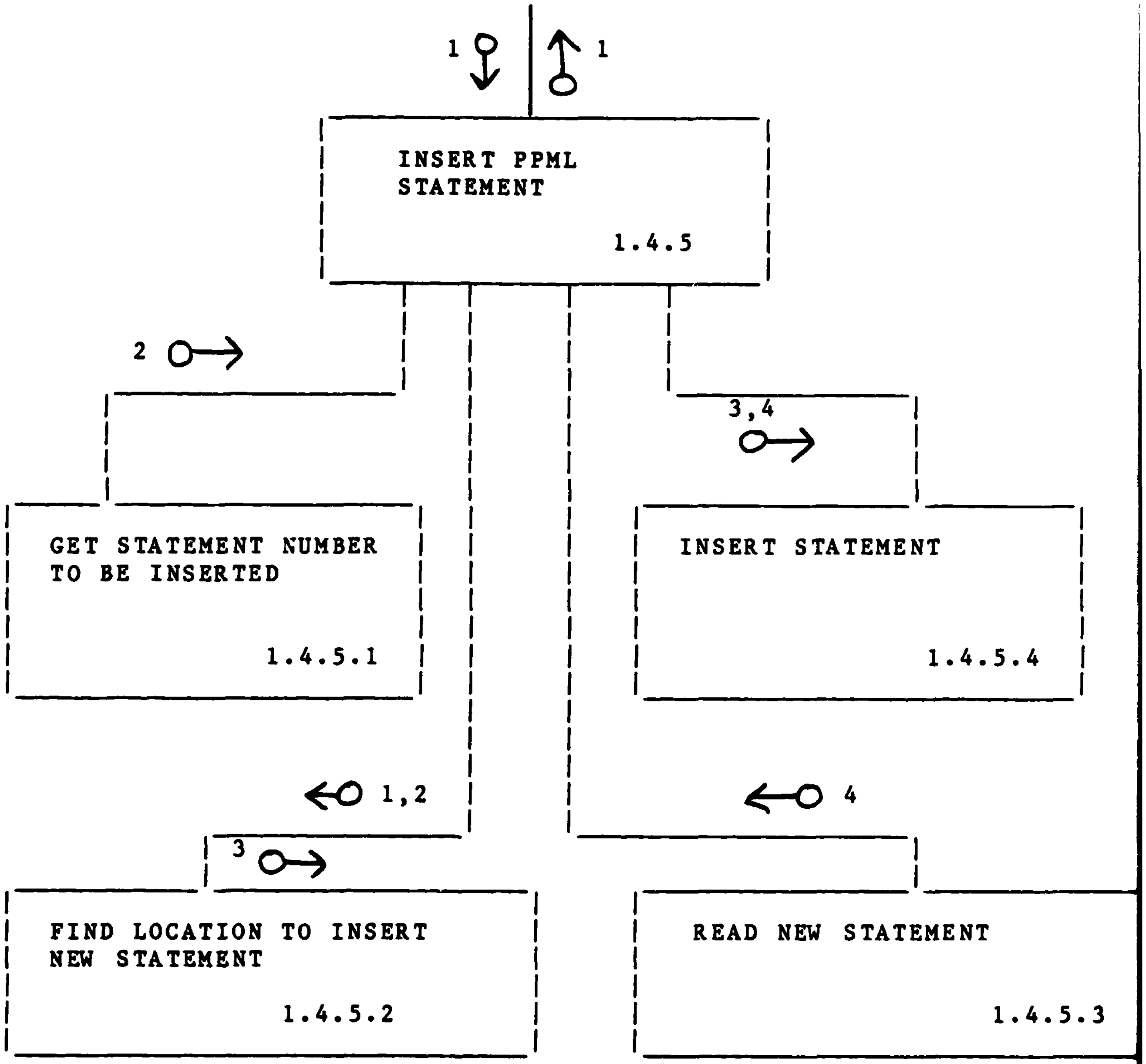
NODE: 1.4.2	TITLE: GLOBAL VARIABLE INPUT	NUMBER:

AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



NODE:	TITLE:	NUMBER:
1.4.3	NEW PPML STATEMENT INPUT	

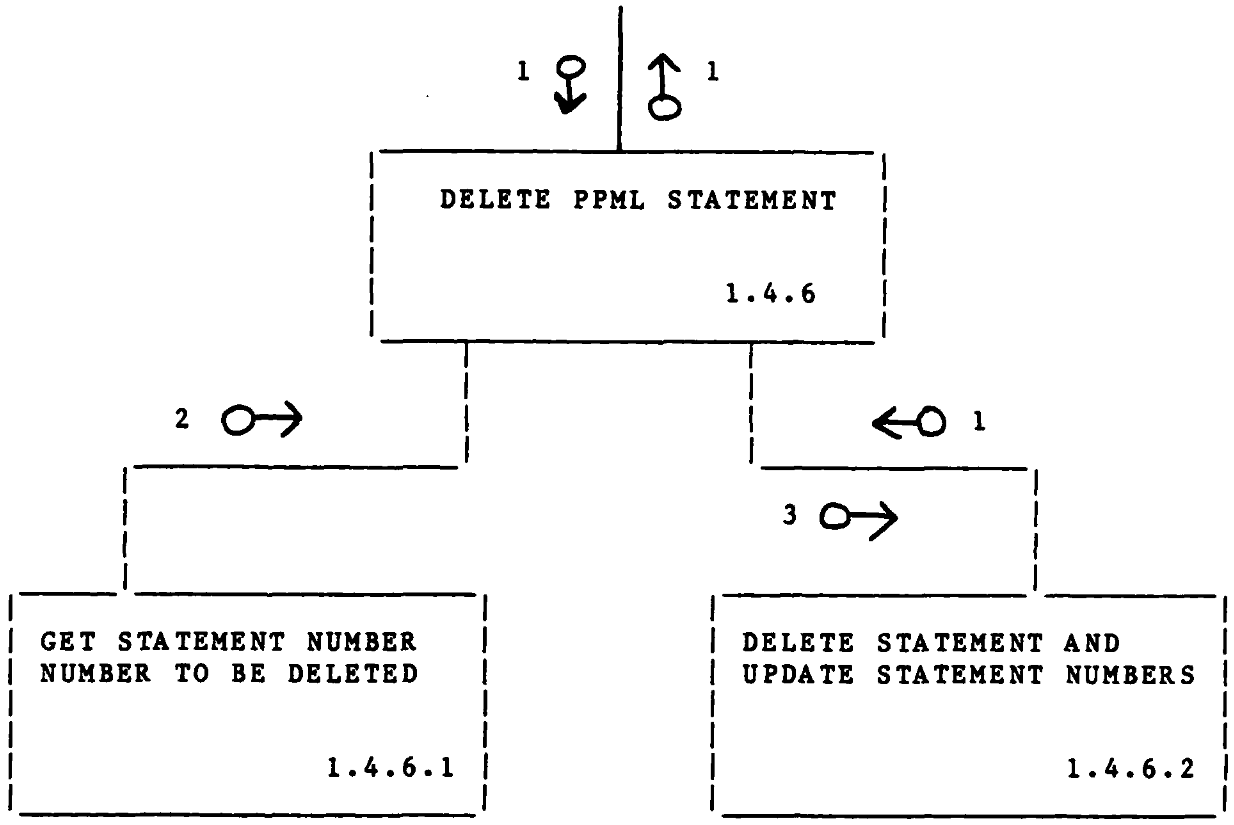
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PROCESS
2. STATEMENT_NUMBER
3. STATEMENT
4. NEW_STATEMENT

NODE: 1.4.5	TITLE: INSERT PPML STATEMENT	NUMBER:

AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83

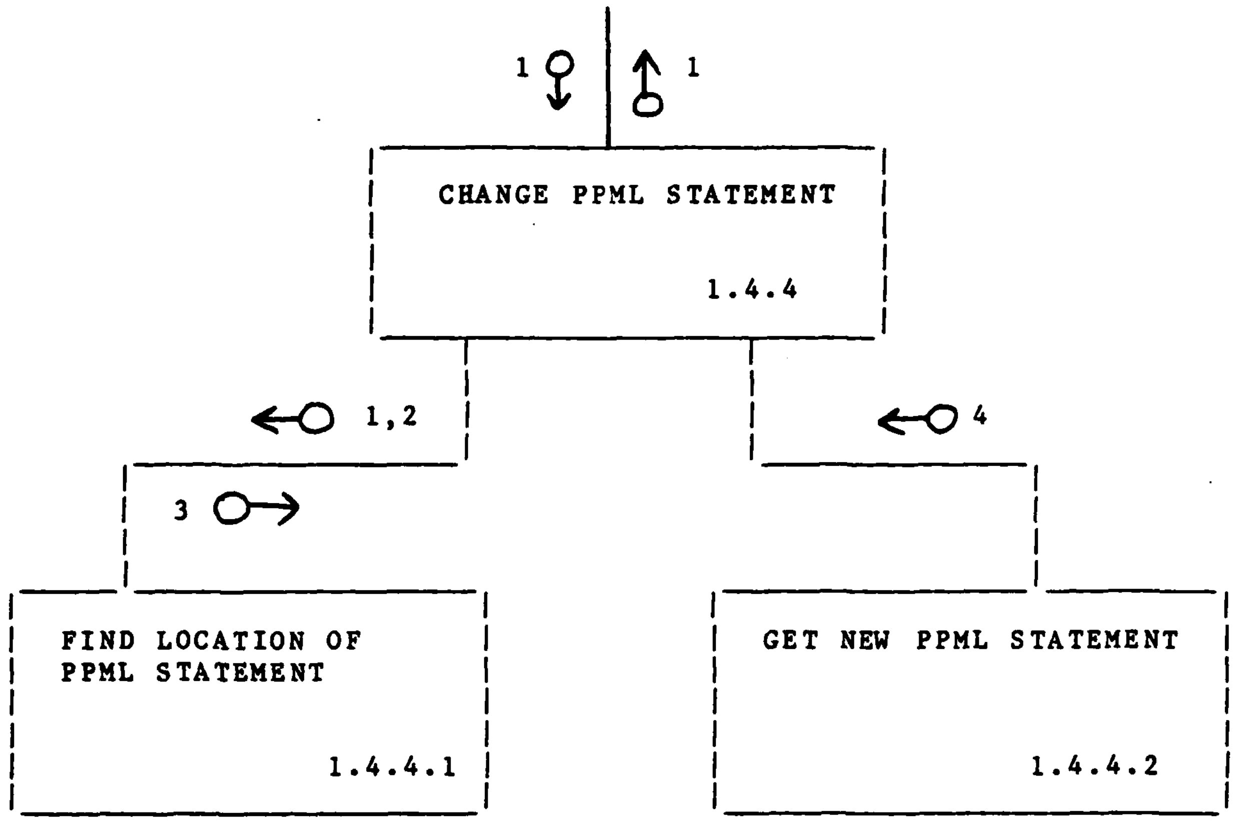


1
1
DELETE PPML STATEMENT
1.4.6
2
1
3
GET STATEMENT NUMBER NUMBER TO BE DELETED
1.4.6.1
DELETE STATEMENT AND UPDATE STATEMENT NUMBERS
1.4.6.2

1. PROCESS
2. STATEMENT_NUMBER
3. STATEMENT_LOCATION

NODE:	TITLE:	NUMBER:
1.4.6	DELETE PPML STATEMENT	

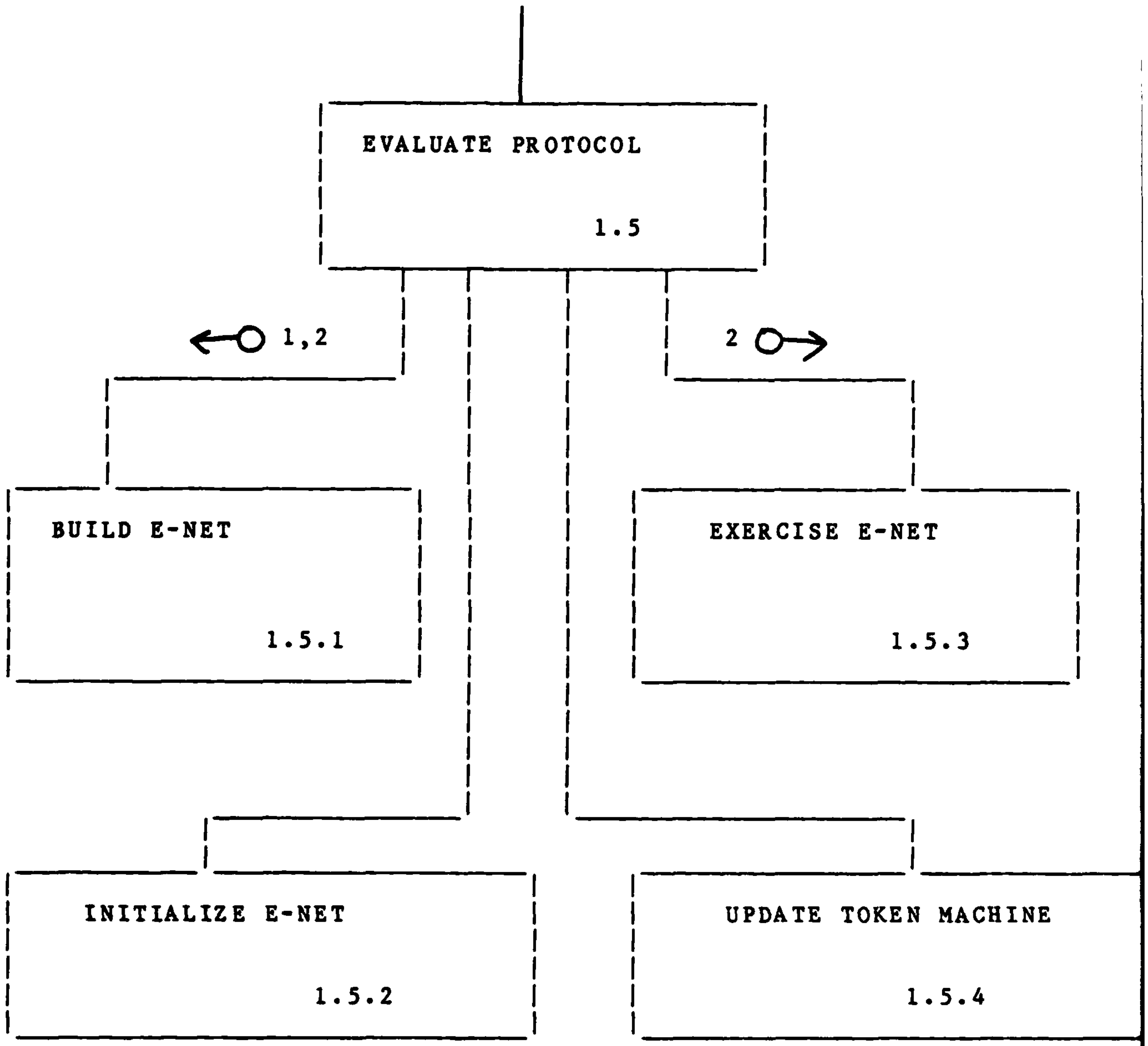
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PROCESS
2. STATEMENT_NUMBER
3. STATEMENT_LOCATION
4. PPML_STATEMENT

NODE: 1.4.4	TITLE: CHANGE PPML STATEMENT	NUMBER:

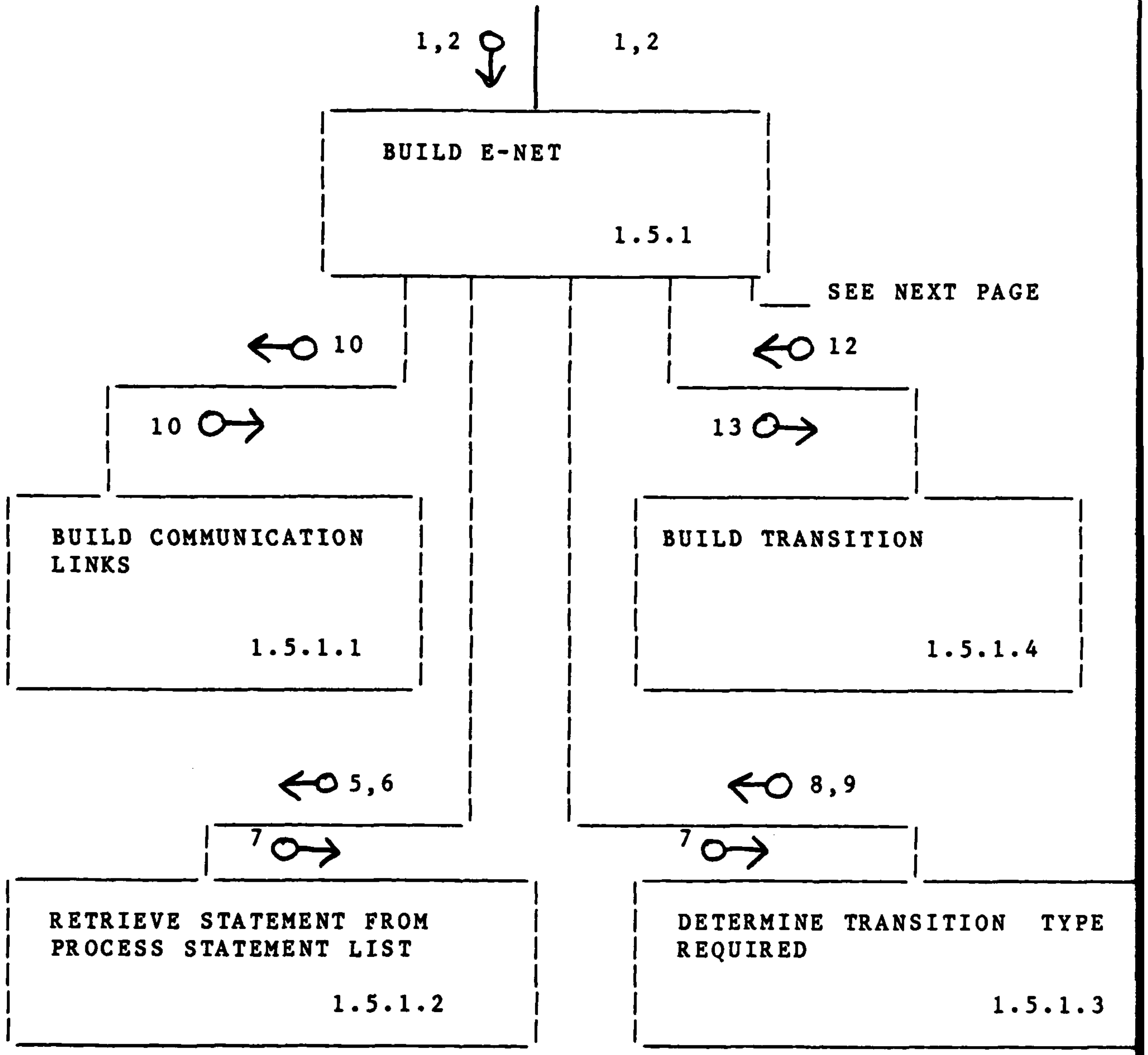
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. NUMBER_OF_PROCESSES
2. PPML_ARRAY

NODE: 1.5	TITLE: EVALUATE PROTOCOL	NUMBER:

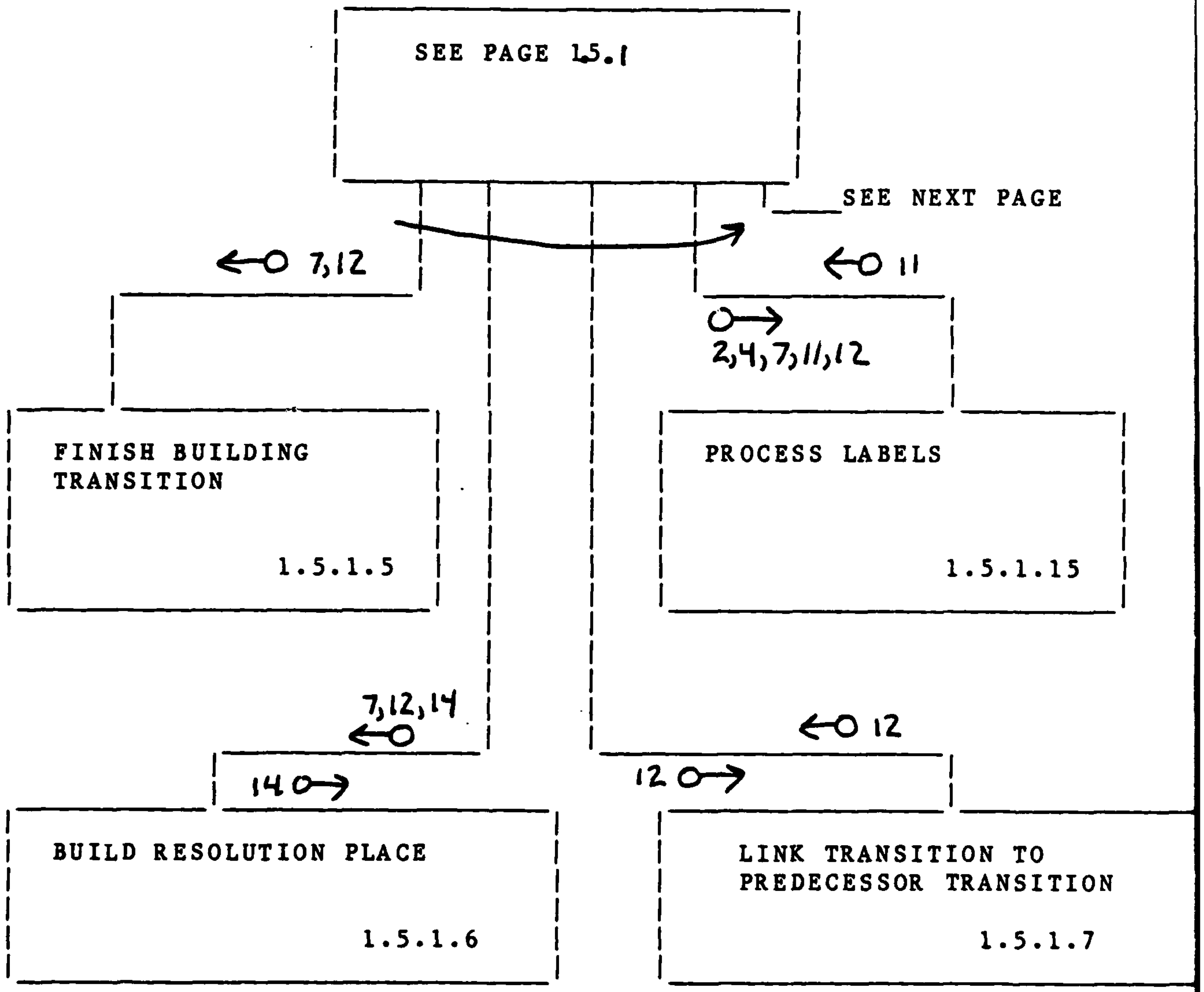
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83


1,2
1,2
BUILD E-NET
1.5.1
SEE NEXT PAGE
10
10
12
13
BUILD COMMUNICATION LINKS
1.5.1.1
BUILD TRANSITION
1.5.1.4
5,6
7
8,9
7
RETRIEVE STATEMENT FROM PROCESS STATEMENT LIST
1.5.1.2
DETERMINE TRANSITION TYPE REQUIRED
1.5.1.3

1. TRANSITION_LIST	9. PPML_TYPE
2. PLACE_LIST	10. LINK_LIST
5. STATEMENT_NUMBER	11. LABEL_LIST
6. PROCESS_NUMBER	12. TRANSITION_POINTER
7. PPML_TYPE	13. TRANSITION_NUMBER
8. TRANSITION_TYPE	

NODE:	TITLE:	NUMBER:
1.5.1	BUILD E-NET (pg. 1 of 3)	

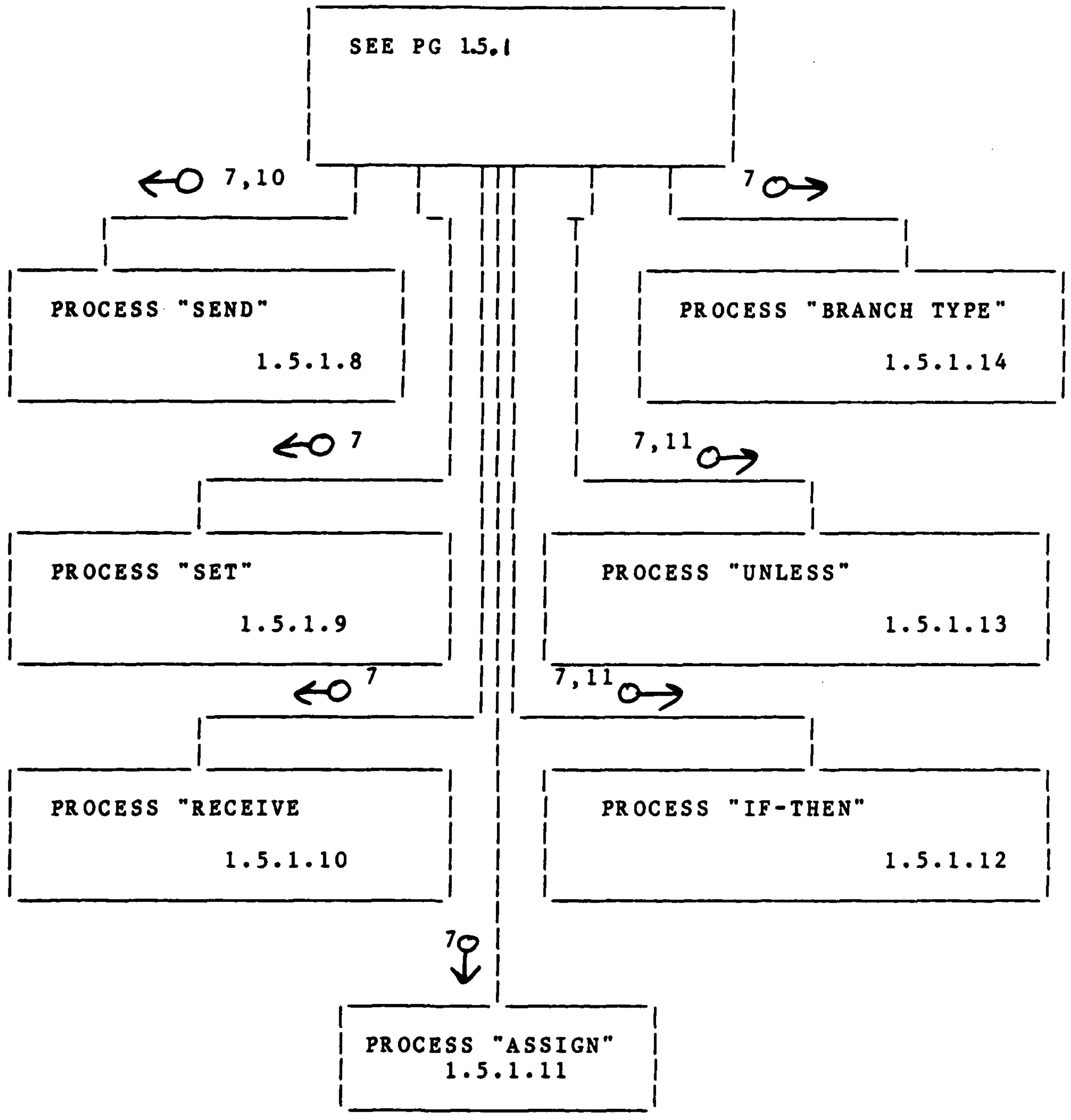
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



SEE PAGE 15 FOR PARAMETER DEFINITIONS

NODE:	TITLE:	NUMBER:
1.5.1	BUILD E-NET (pg. 2 of 3)	

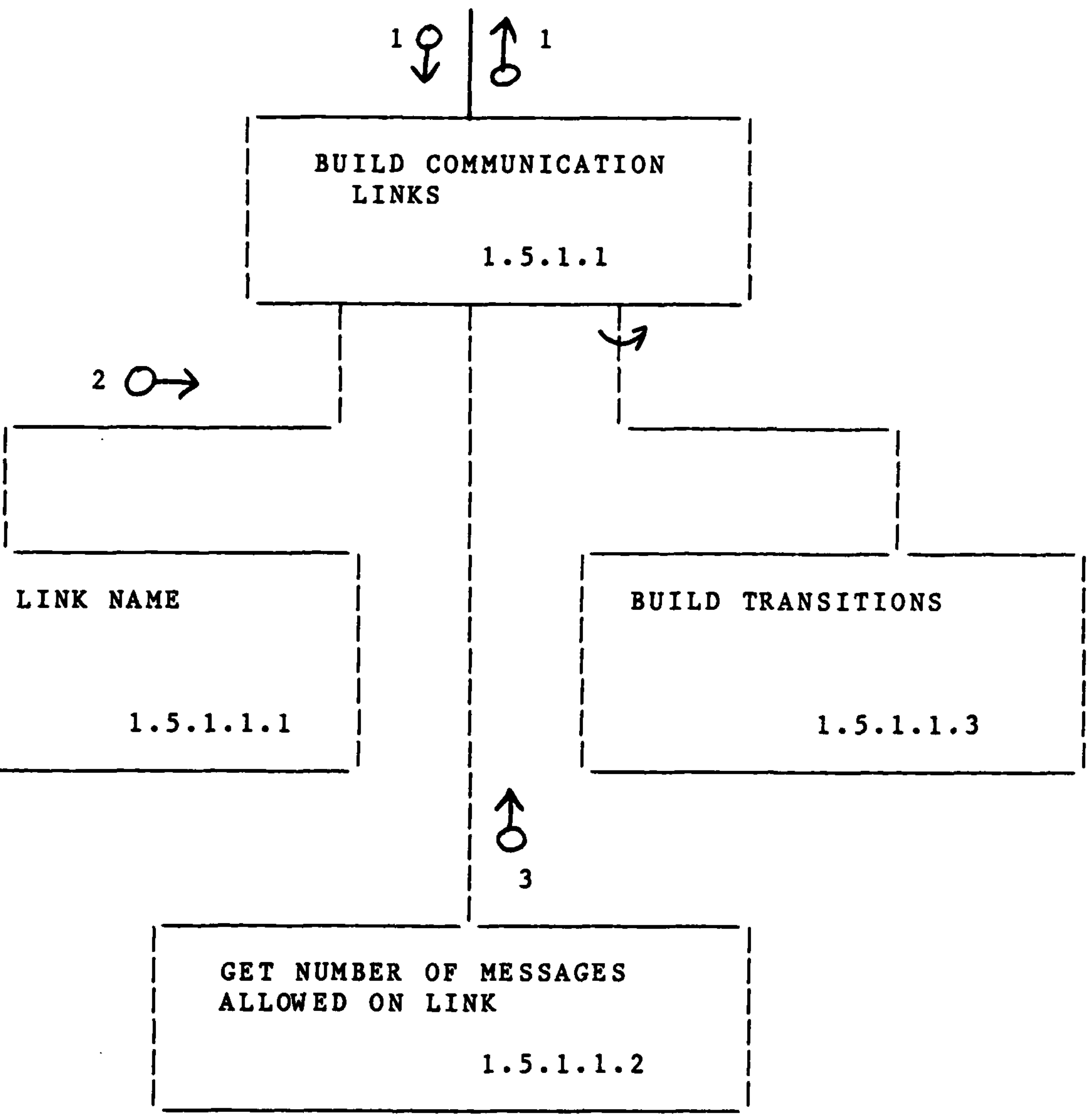
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83


SEE PG 1.5.1
7,10
7
PROCESS "SEND" 1.5.1.8
PROCESS "BRANCH TYPE" 1.5.1.14
7
7,11
PROCESS "SET" 1.5.1.9
PROCESS "UNLESS" 1.5.1.13
7
7,11
PROCESS "RECEIVE 1.5.1.10
PROCESS "IF-THEN" 1.5.1.12
7
PROCESS "ASSIGN" 1.5.1.11

SEE PG 15 FOR PARAMETER DEFINITIONS

NODE: 1.5.1	TITLE: BUILD E-NET (pg 3 of 3)	NUMBER:

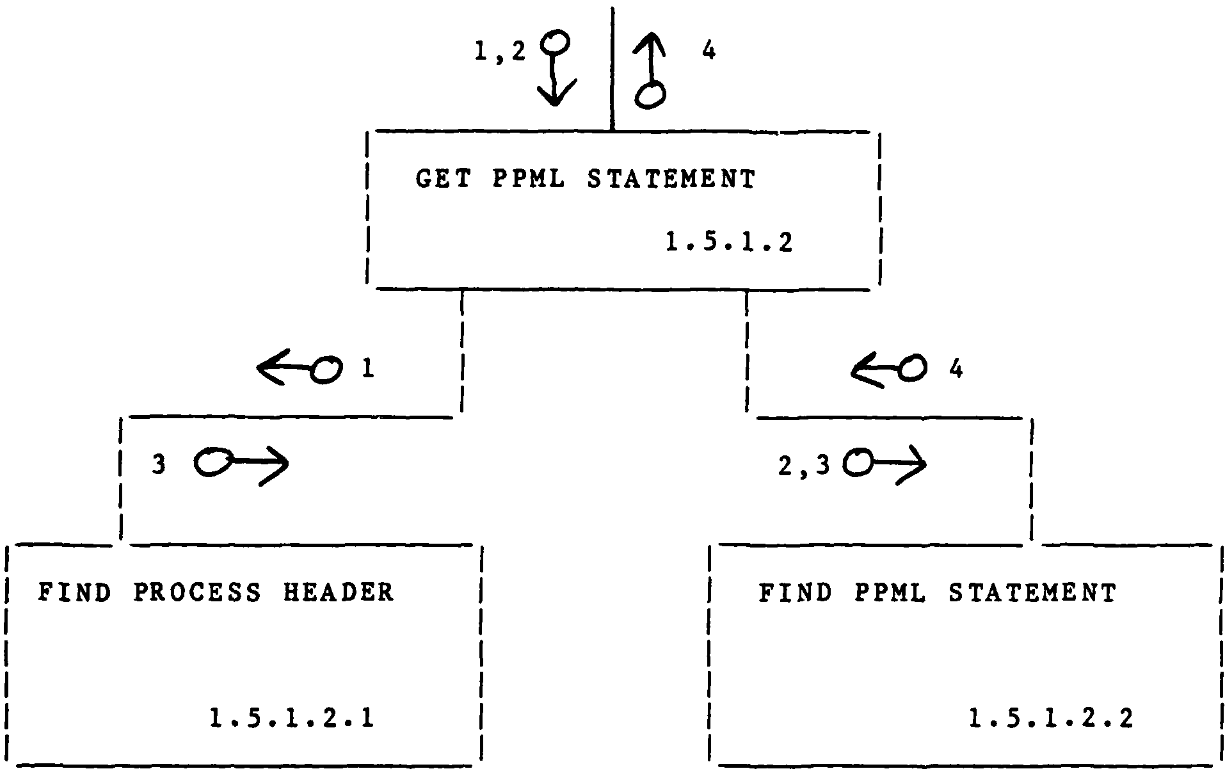
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. LINK_LIST
2. LINK_NAME
3. NUMBER_OF_MESSAGES

NODE:	TITLE:	NUMBER:
1.5.1.1	BUILD COMMUNICATION LINKS	

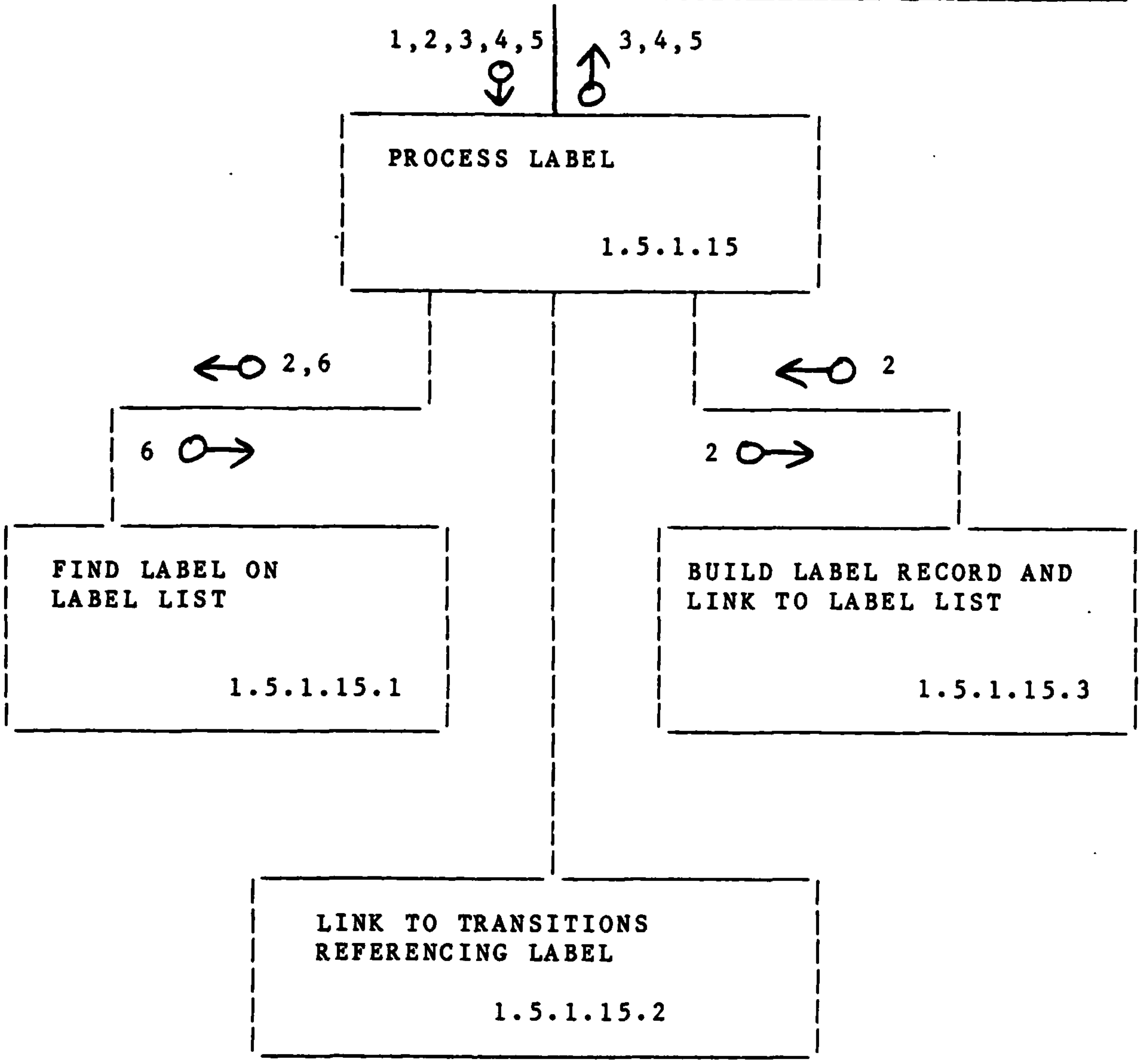
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PROCESS_NUMBER
2. STATEMENT_NUMBER
3. PROCESS_POINTER
4. PPML_STATEMENT_POINTER

NODE: 1.5.1.2	TITLE: GET PPML STATEMENT	NUMBER:

AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83

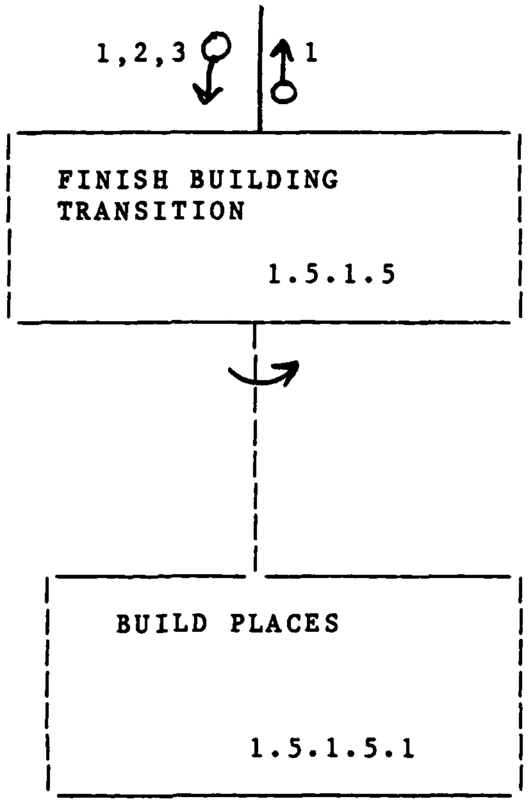


1,2,3,4,5
3,4,5
PROCESS LABEL 1.5.1.15
2,6
6
2
2
FIND LABEL ON LABEL LIST 1.5.1.15.1
BUILD LABEL RECORD AND LINK TO LABEL LIST 1.5.1.15.3
LINK TO TRANSITIONS REFERENCING LABEL 1.5.1.15.2

1. PPML_STATEMENT
2. LABEL_LIST
3. TRANSITION_LIST
4. PLACE_LIST
5. RESOLUTION_PLACE_LIST
6. LABEL
7. LABEL_PTR

NODE: 1.5.1.15	TITLE: PROCESS LABEL	NUMBER:

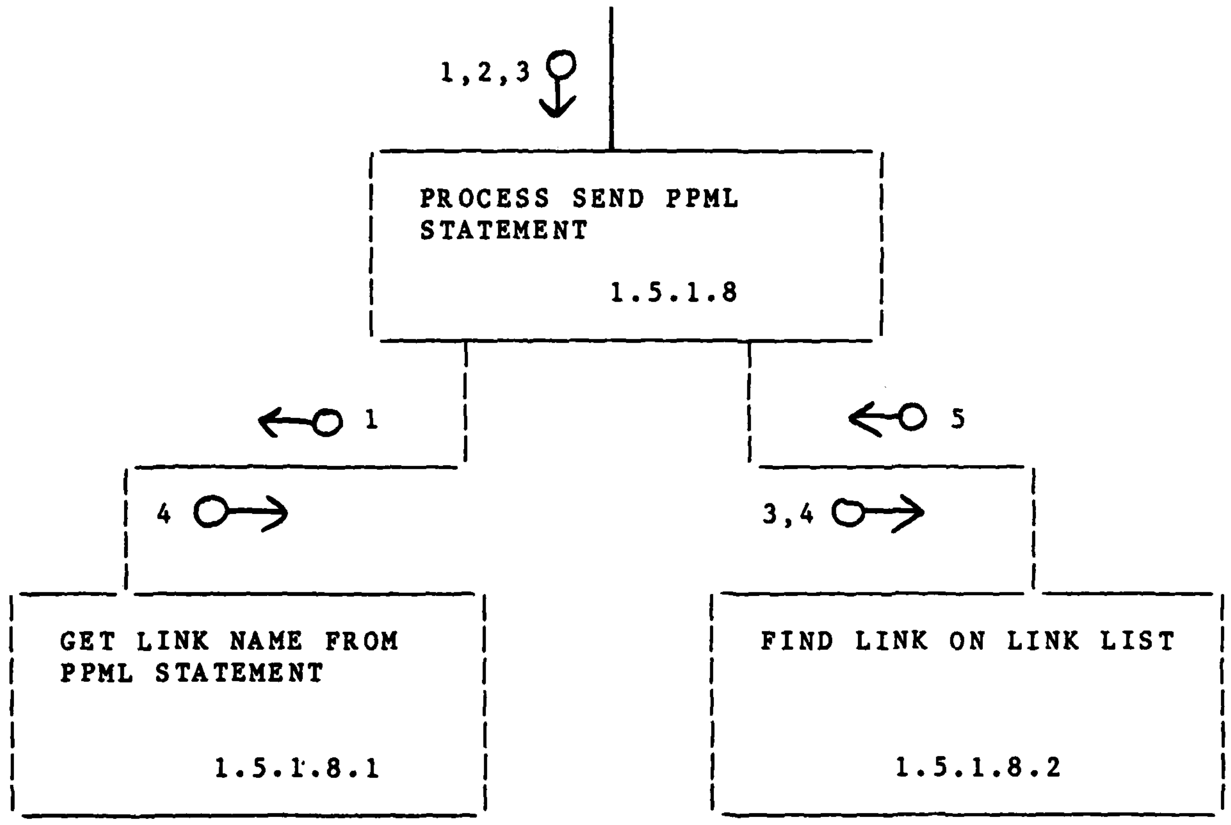
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. TRANSITION
2. TRANSITION_TYPE
3. PPML_TYPE

NODE: 1.5.1.5	TITLE: FINISH BUILDING TRANSTION	NUMBER:

AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83

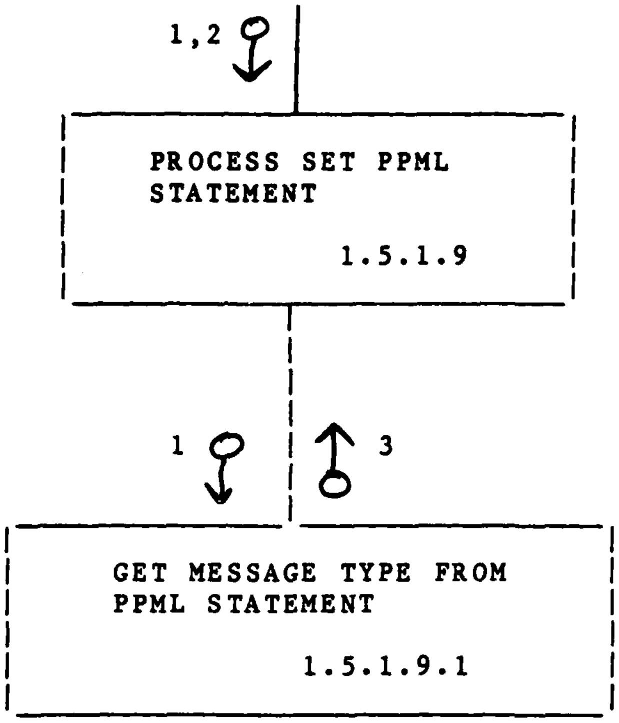


1,2,3
PROCESS SEND PPML STATEMENT
1.5.1.8
1
5
4
3,4
GET LINK NAME FROM PPML STATEMENT
1.5.1.8.1
FIND LINK ON LINK LIST
1.5.1.8.2

1. PPML_STATEMENT
2. SEND_TRANSITIOMN
3. LINK_LIST
4. LINK_NAME
5. LINK_PTR

NODE: 1.5.1.8	TITLE: PROCESS SEND PPML STATEMENT	NUMBER:

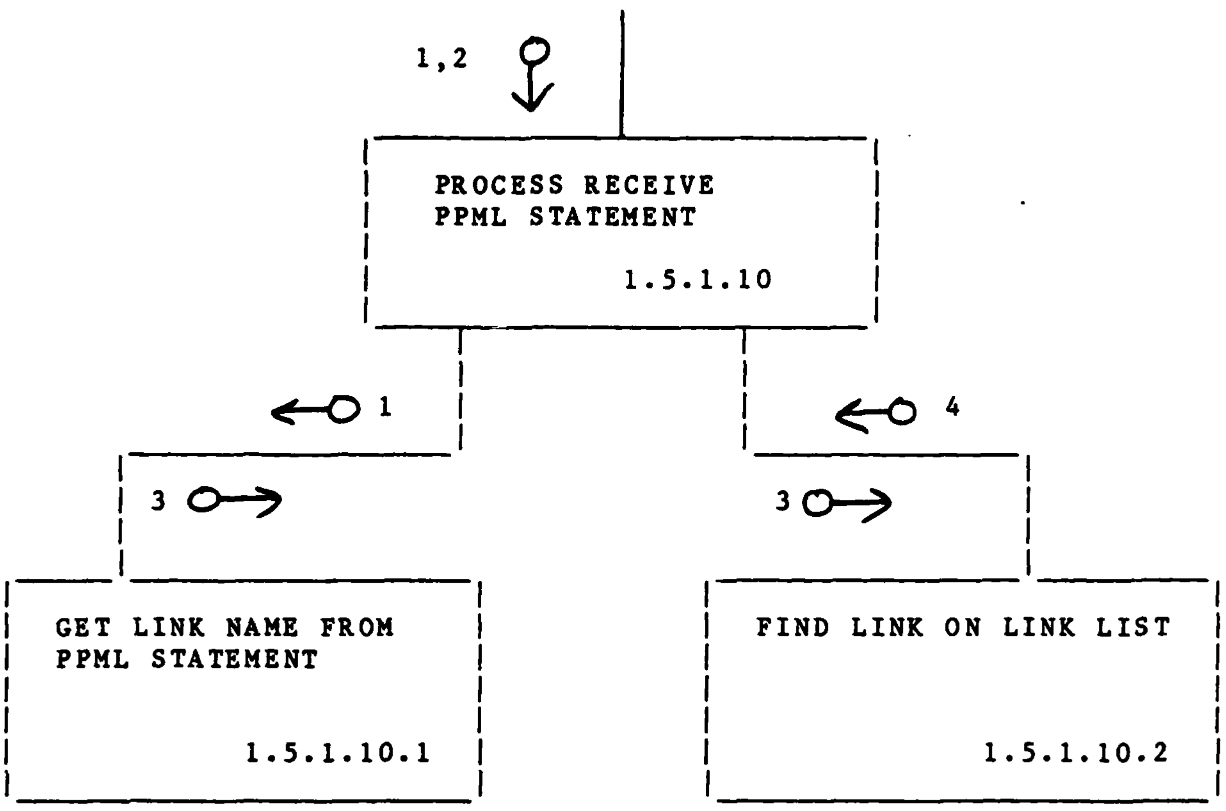
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83


1,2
PROCESS SET PPML STATEMENT
1.5.1.9
1
3
GET MESSAGE TYPE FROM PPML STATEMENT
1.5.1.9.1

1. PPML_STATEMENT
2. SET_TRANSITION
3. LINK_NAME

NODE: 1.5.1.9	TITLE: PROCESS SET PPLML STATEMENT	NUMBER:

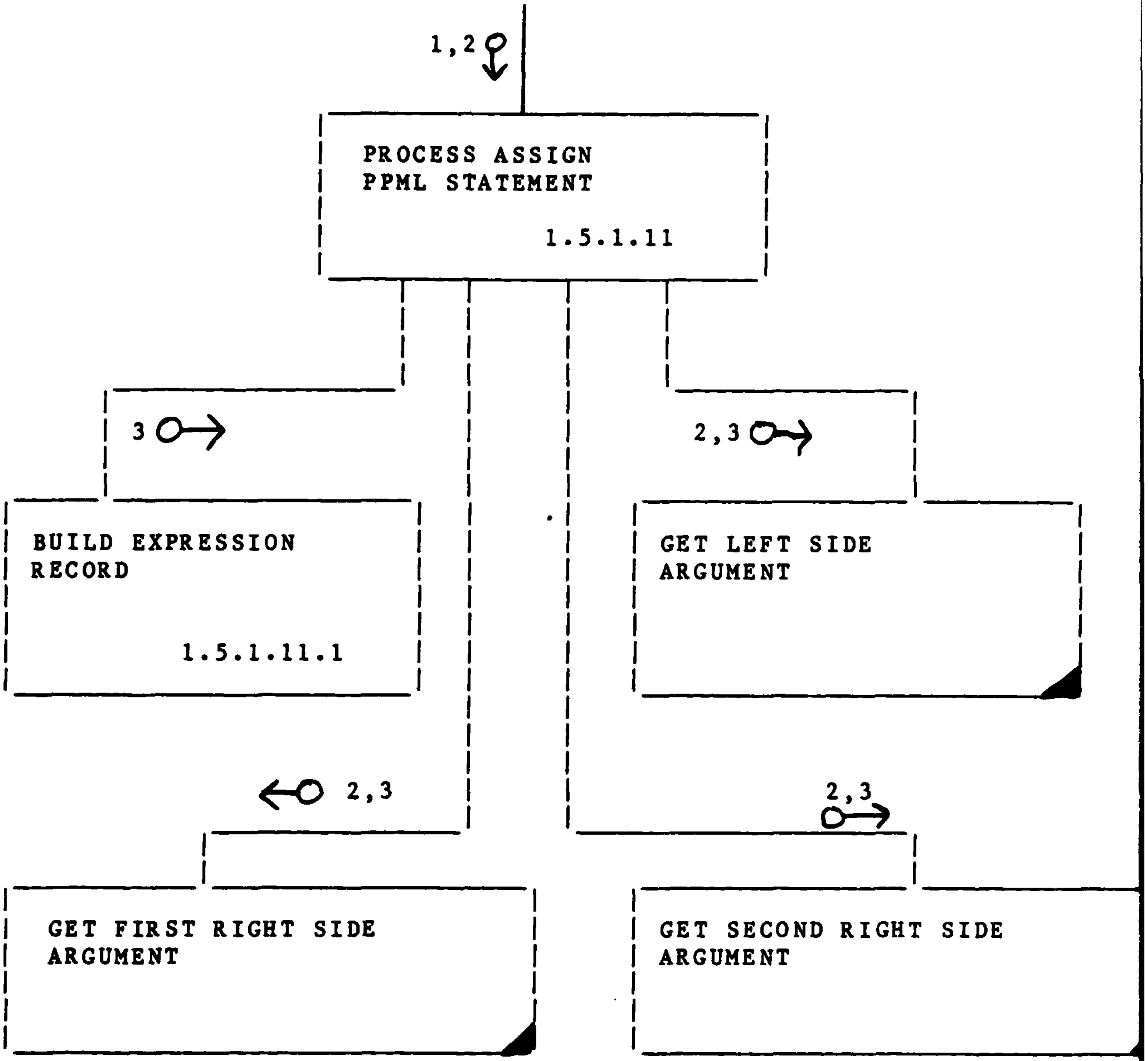
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83


1,2
PROCESS RECEIVE PPML STATEMENT
1.5.1.10
1
4
3
3
GET LINK NAME FROM PPML STATEMENT
1.5.1.10.1
FIND LINK ON LINK LIST
1.5.1.10.2

1. PPML_STATEMENT
2. RECEIVE_TRANSITION
3. LINK_NAME
4. LINK_PTR

NODE:	TITLE:	NUMBER:
1.5.1.10	PROCESS RECEIVE PPML STATEMENT	

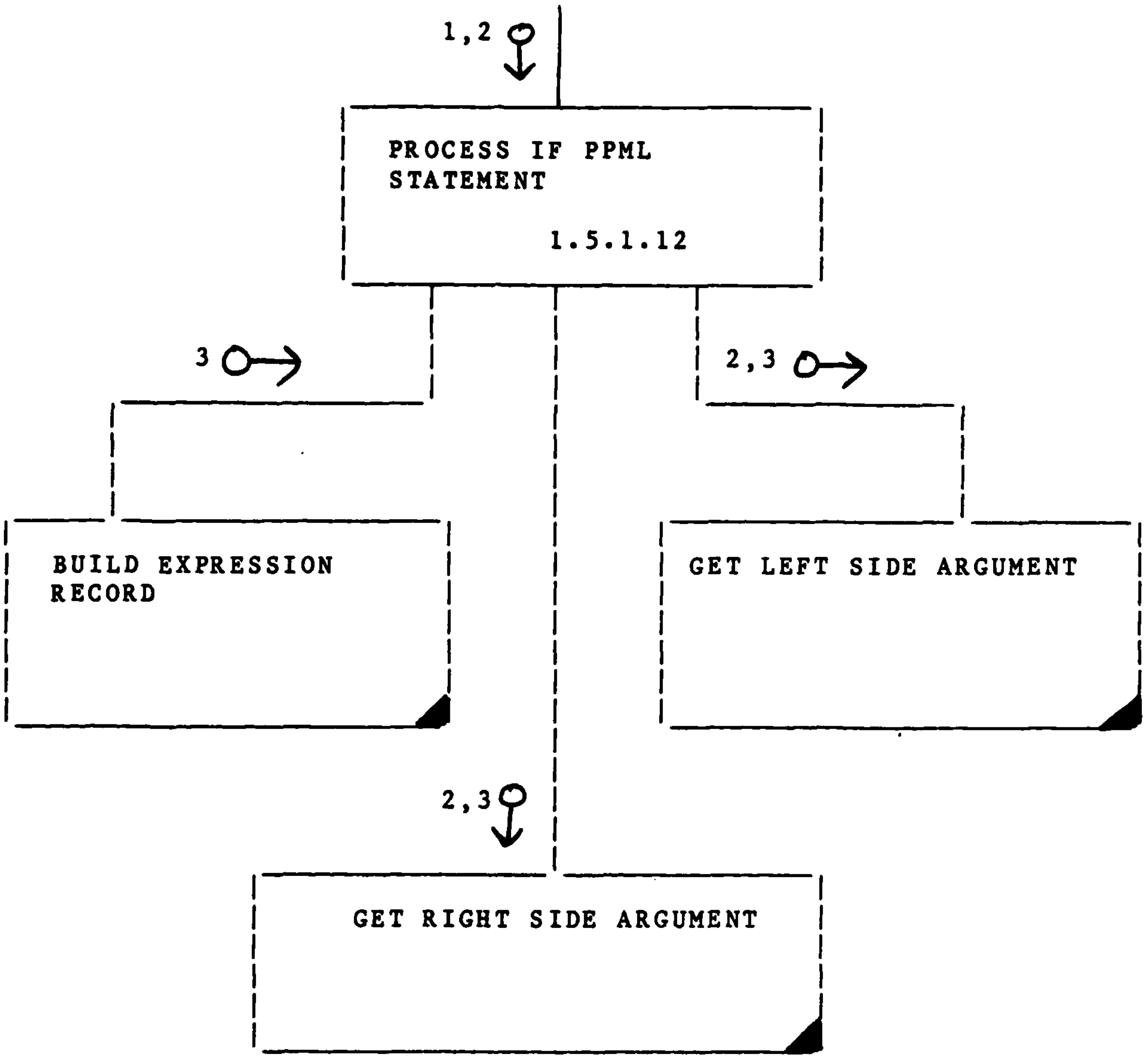
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83


1,2
PROCESS ASSIGN PPML STATEMENT
1.5.1.11
3
2,3
BUILD EXPRESSION RECORD
1.5.1.11.1
GET LEFT SIDE ARGUMENT
2,3
2,3
GET FIRST RIGHT SIDE ARGUMENT
GET SECOND RIGHT SIDE ARGUMENT

1. PPML_STATEMENT
2. TRANSITION
3. EXPRESSION_RECORD

NODE:	TITLE:	NUMBER:
1.5.1.11	PROCESS ASSIGN PPML STATEMENT	

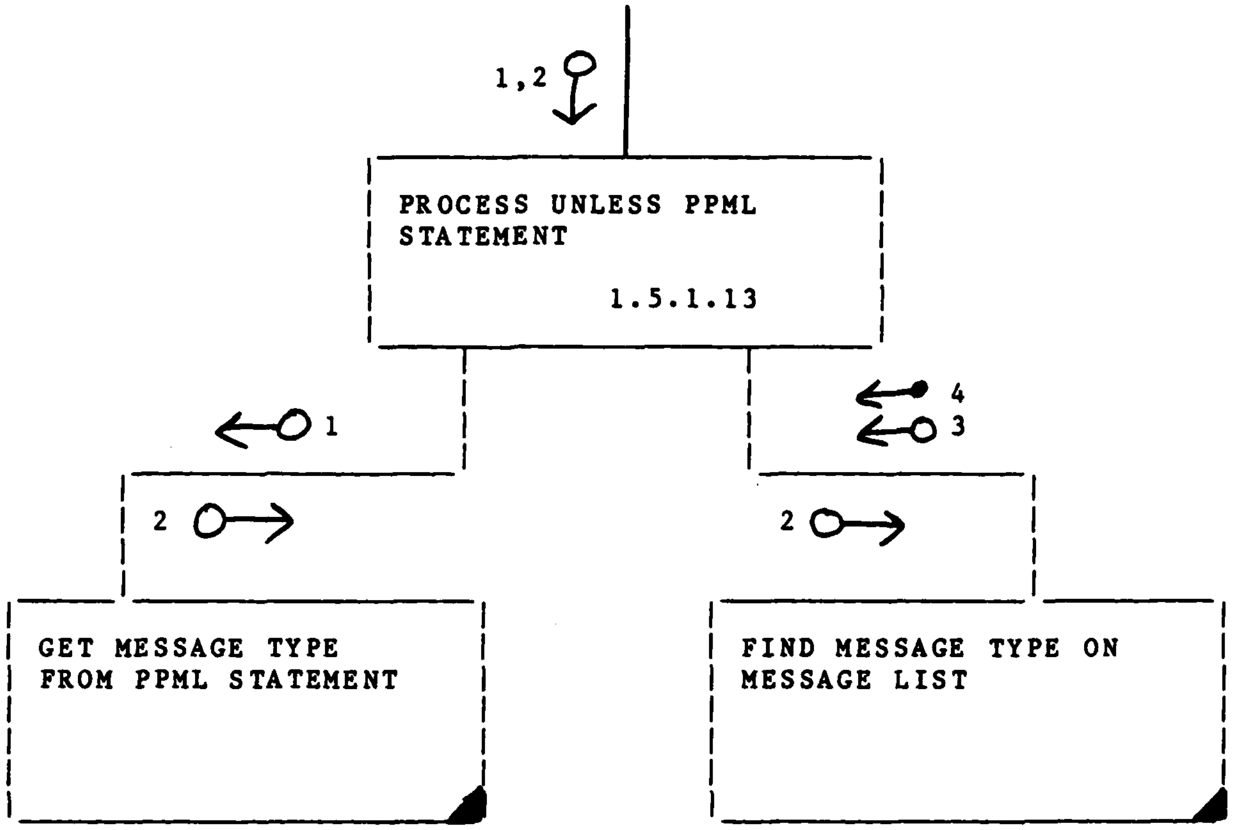
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PPML_STATEMENT
2. TRANSITION
3. EXPRESSION_RECORD

NODE: 1.5.1.12	TITLE: PROCESS IF PPML STATEMENT	NUMBER:

AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83

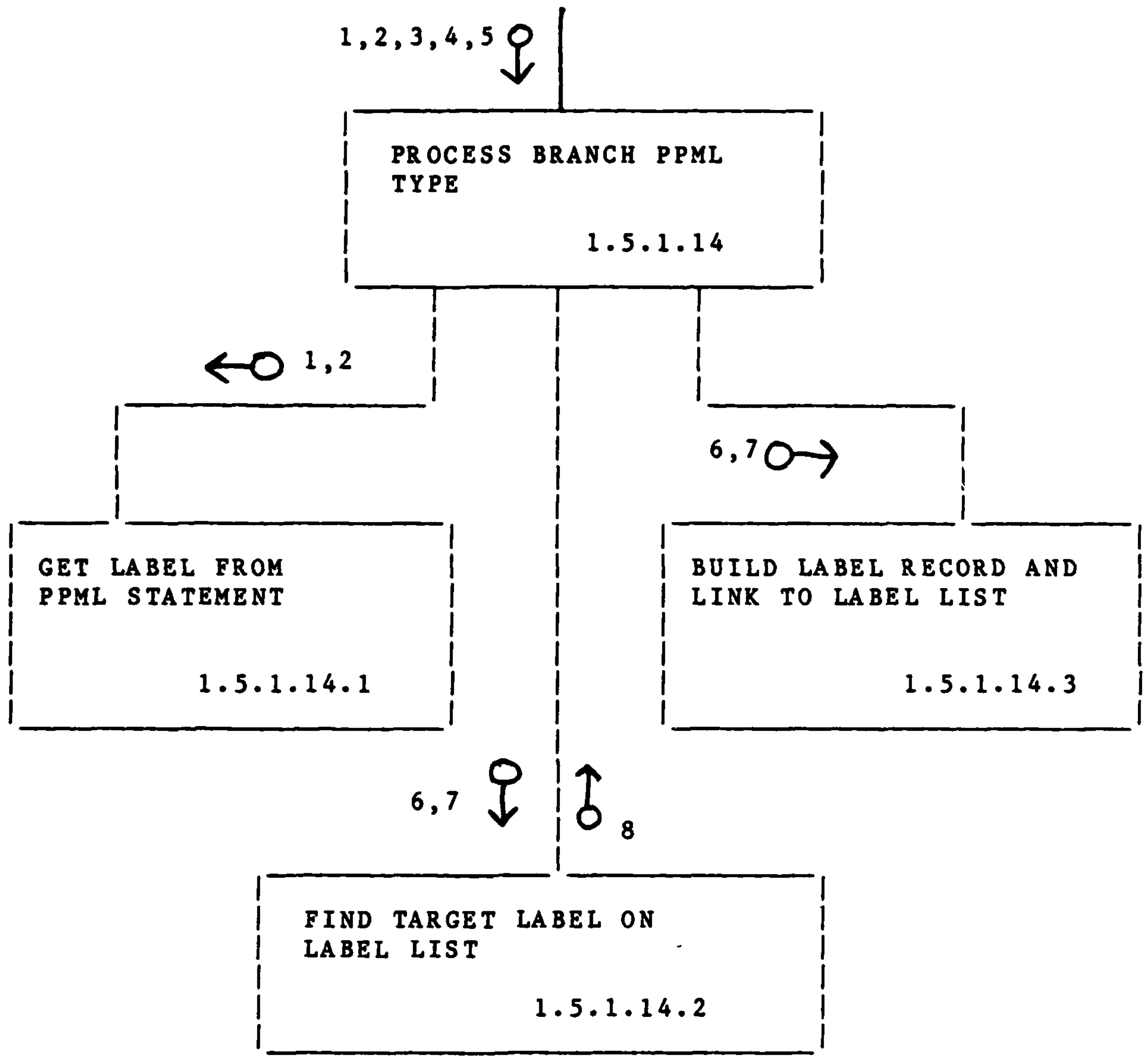


1,2
PROCESS UNLESS PPML STATEMENT
1.5.1.13
1
4
3
2
2
GET MESSAGE TYPE FROM PPML STATEMENT
FIND MESSAGE TYPE ON MESSAGE LIST

1. TRANSITION
2. MESSAGE_TYPE
3. MESSAGE_PTR
4. FOUND_FLAG

NODE: 1.5.1.13	TITLE: PROCESS ASSIGN PPML STATEMENT	NUMBER:

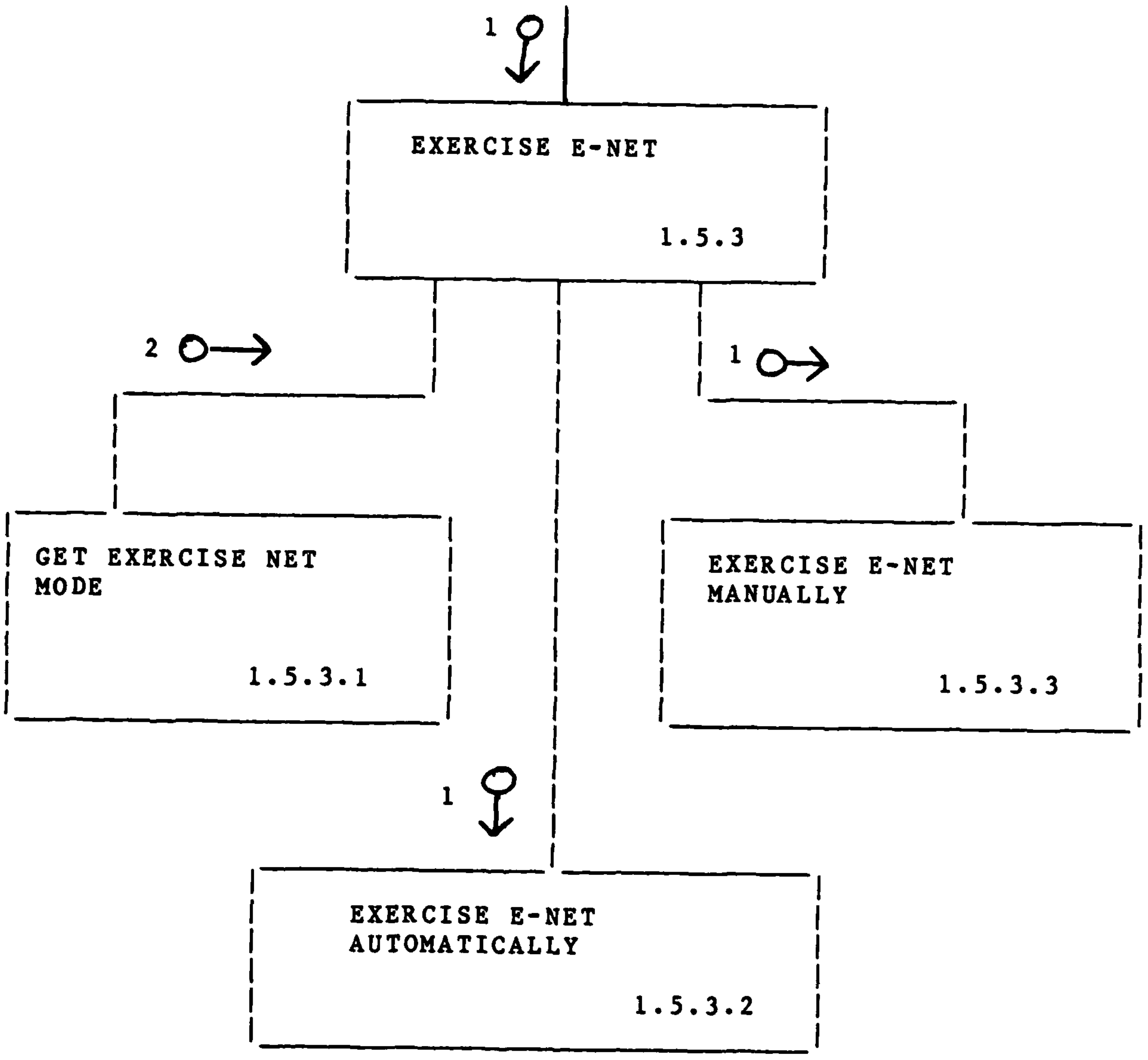
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83


1,2,3,4,5
PROCESS BRANCH PPML TYPE
1.5.1.14
1,2
6,7
GET LABEL FROM PPML STATEMENT
1.5.1.14.1
BUILD LABEL RECORD AND LINK TO LABEL LIST
1.5.1.14.3
6,7
8
FIND TARGET LABEL ON LABEL LIST
1.5.1.14.2

1. PPML_STATEMENT	6. LABEL
2. TRANSITION	7. LABEL_LIST
3. TRRANSITION_LIST	8. LABEL_PTR
4. PLACE_LIST	
5. RESOLUTION_PLACE_LIST	

NODE:	TITLE:	NUMBER:
1.5.1.14	PROCESS BRANCH PPML TYPE	

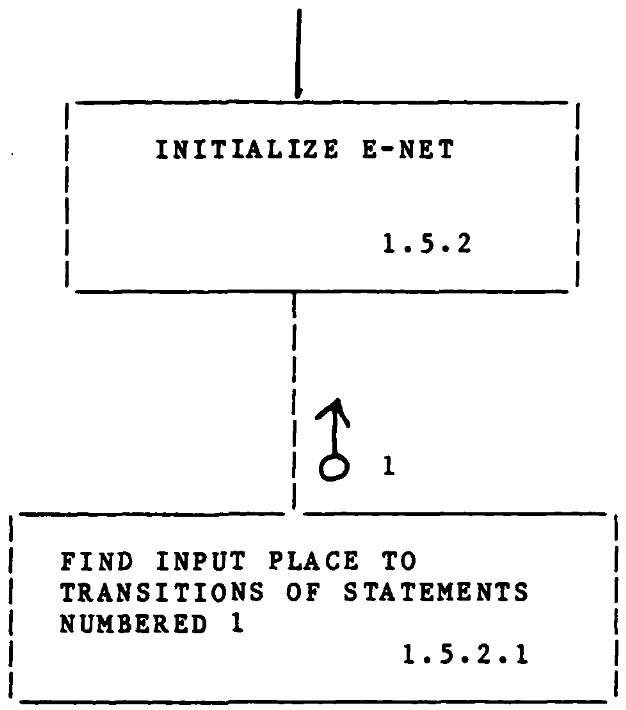
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PPML_ARRAY
2. EXERCISE_MODE

NODE: 1.5.3	TITLE: EXERCISE E-NET	NUMBER:

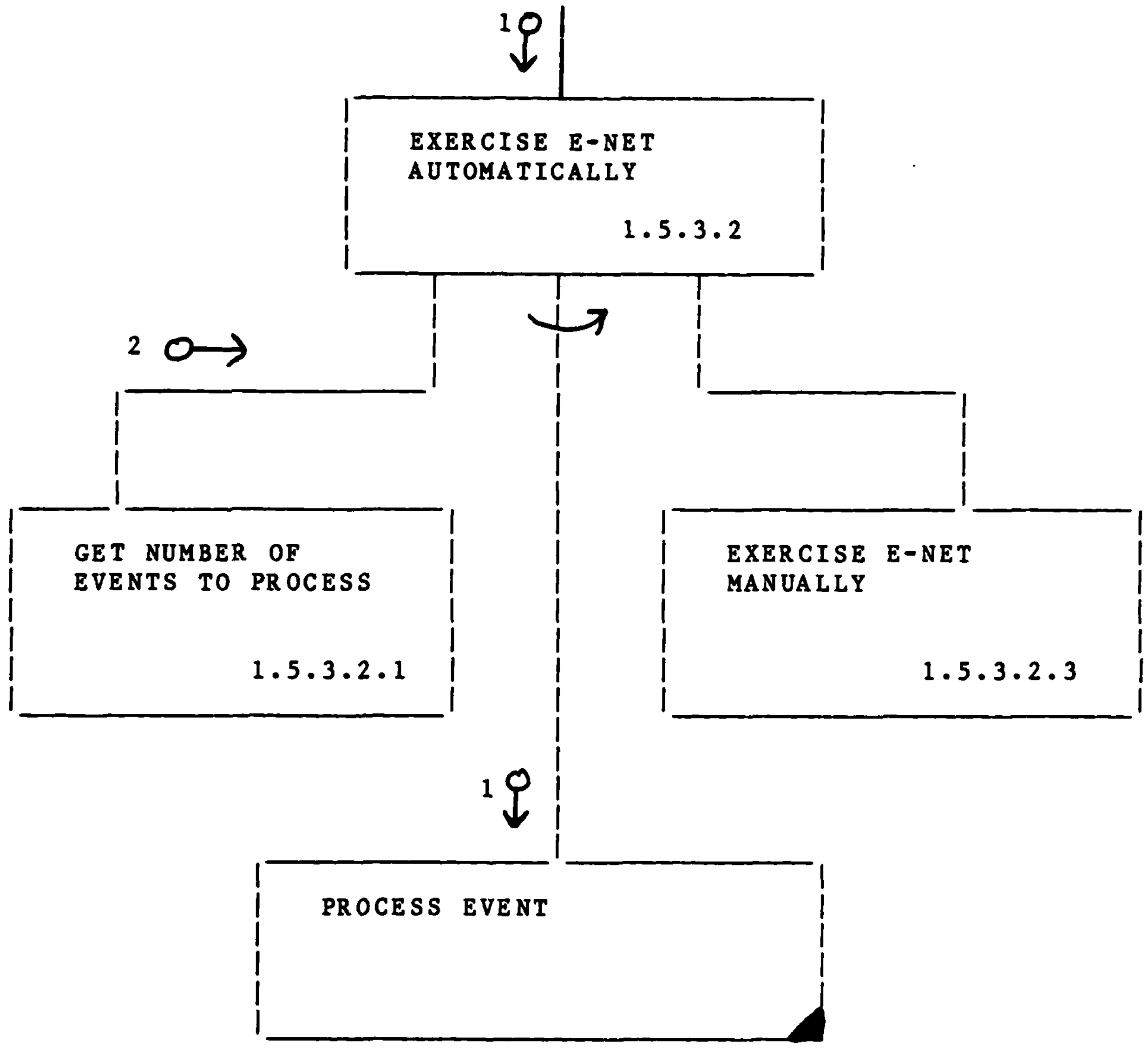
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. PLACE_PTR

NODE:	TITLE:	NUMBER:
1.5.2	INITIALIZE E-NET	

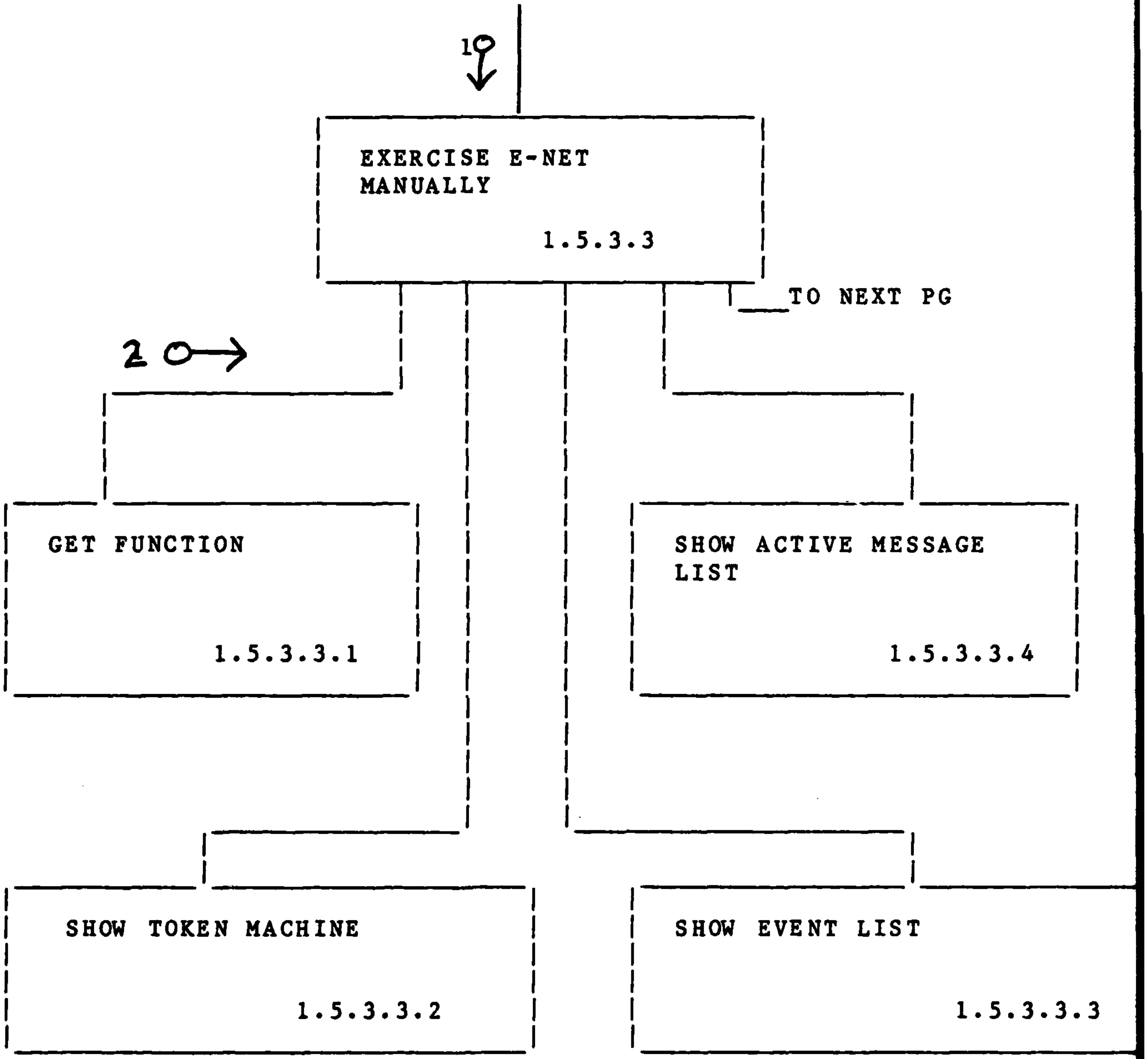
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PPML_ARRAY
2. NUMBER_OF_EVENTS

NODE: 1.5.3.2	TITLE: EXERCISE E-NET AUTOMATICALLY	NUMBER:

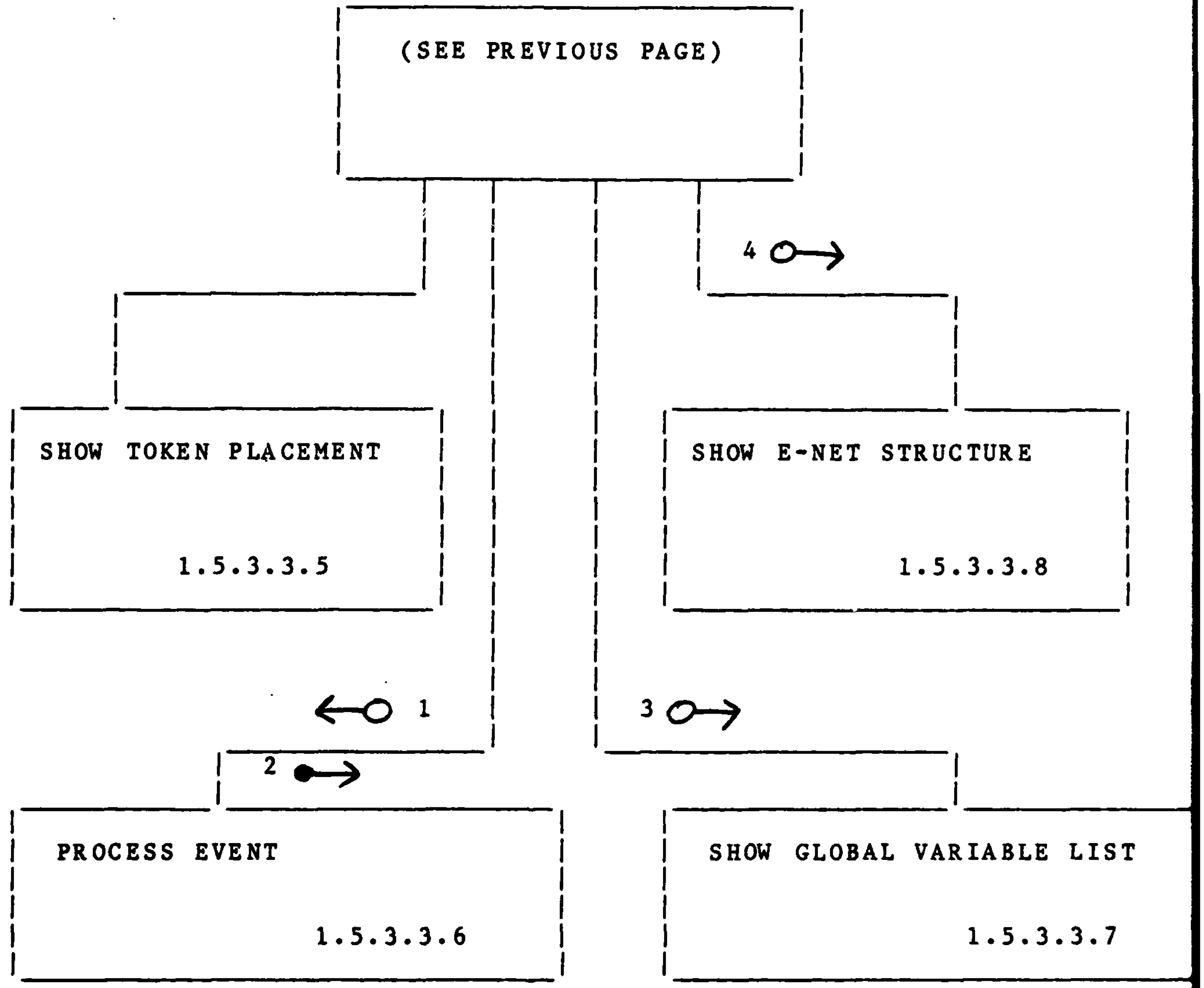
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. PPML_ARRAY
2. FUNCTION

NODE:	TITLE:	NUMBER:
1.5.3.3	EXERCISE E-NET MANUALLY(1 of 2)	

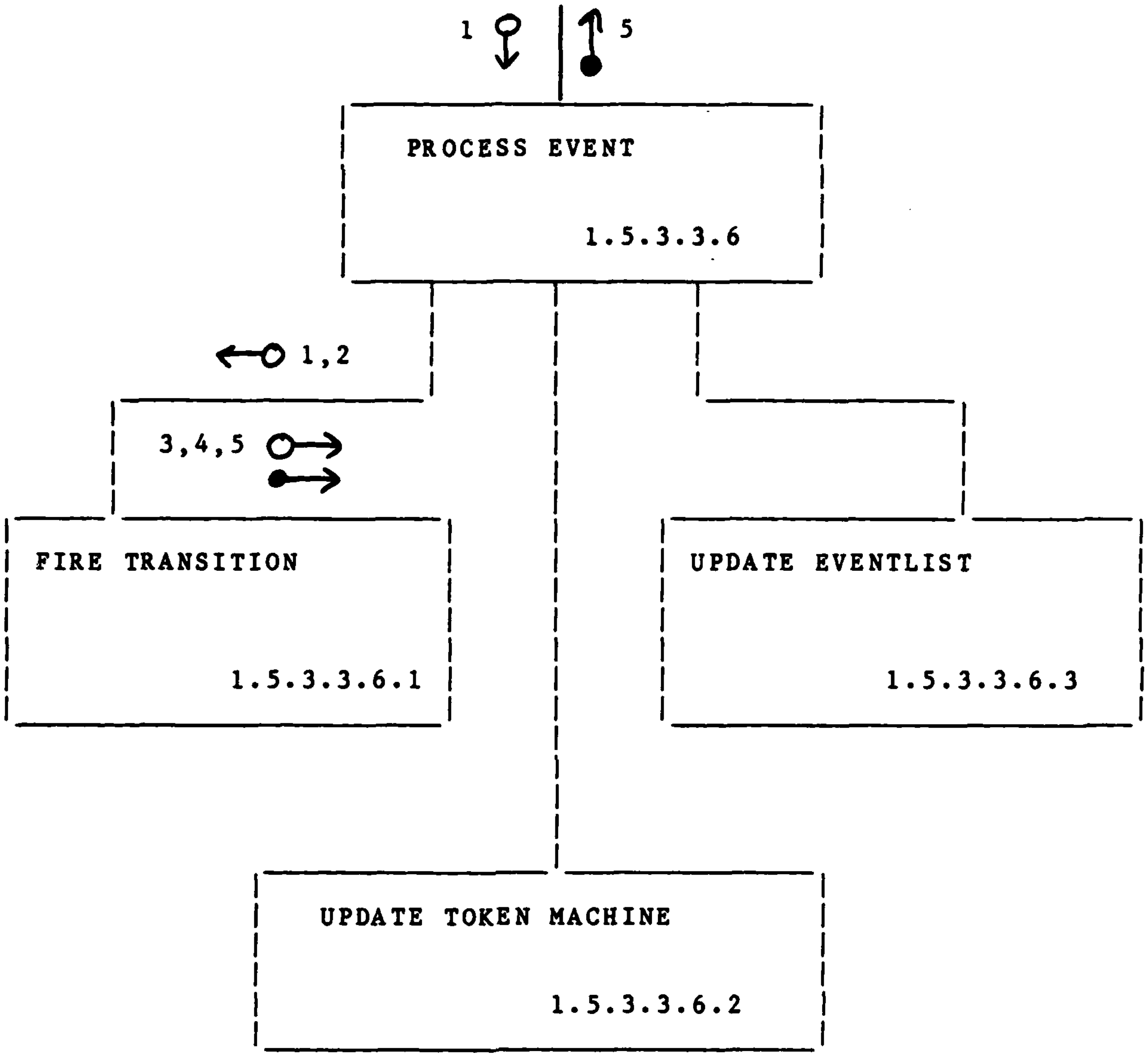
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. PPML_ARRAY
2. ERROR_FLAG
3. GLOBAL_VARIABLE_LIST
4. TRANSITION_LIST

NODE: 1.5.3.3	TITLE: EXERCISE E-NET MANUALLY(2 of 2)	NUMBER:

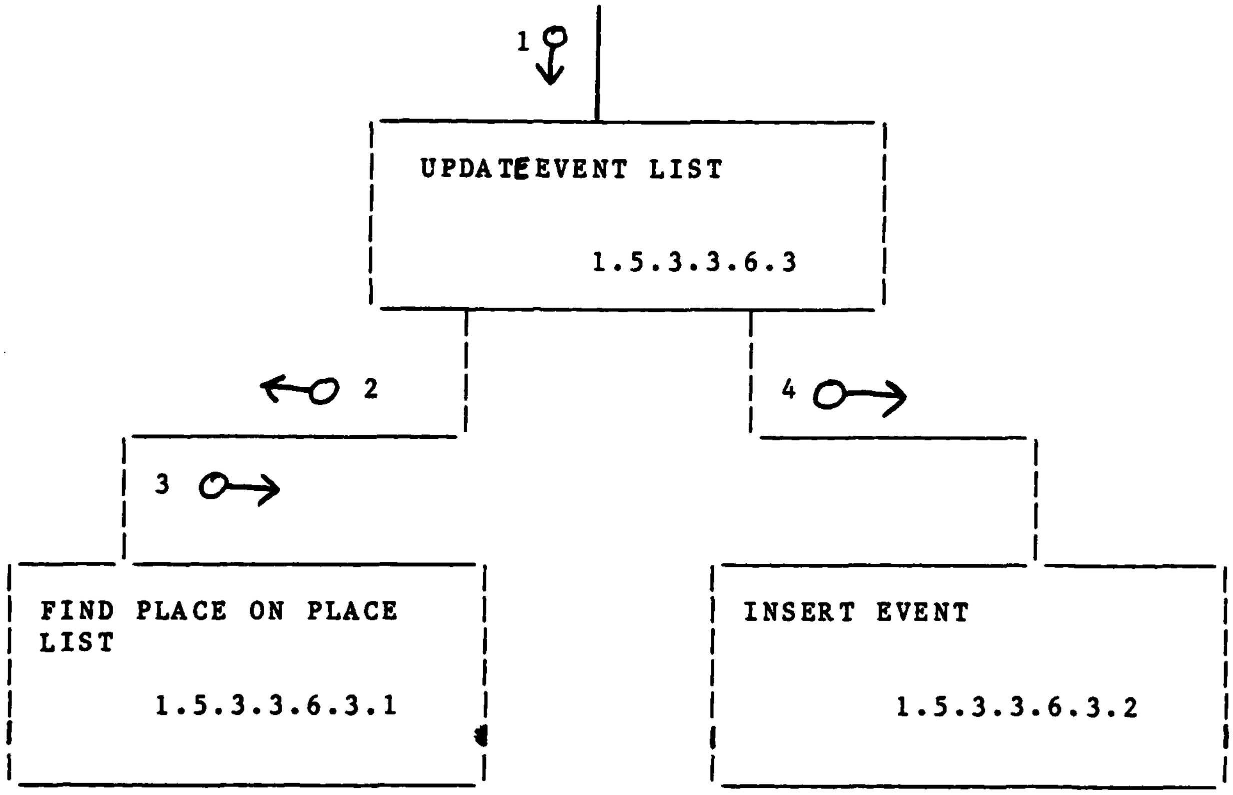
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. PPML_ARRAY
2. TRANSITION
3. OUTPLACE_1
4. OUTPLACE_2
5. ERROR_FLAG

NODE:	TITLE:	NUMBER:
1.5.3.3.6	PROCESS EVENT	

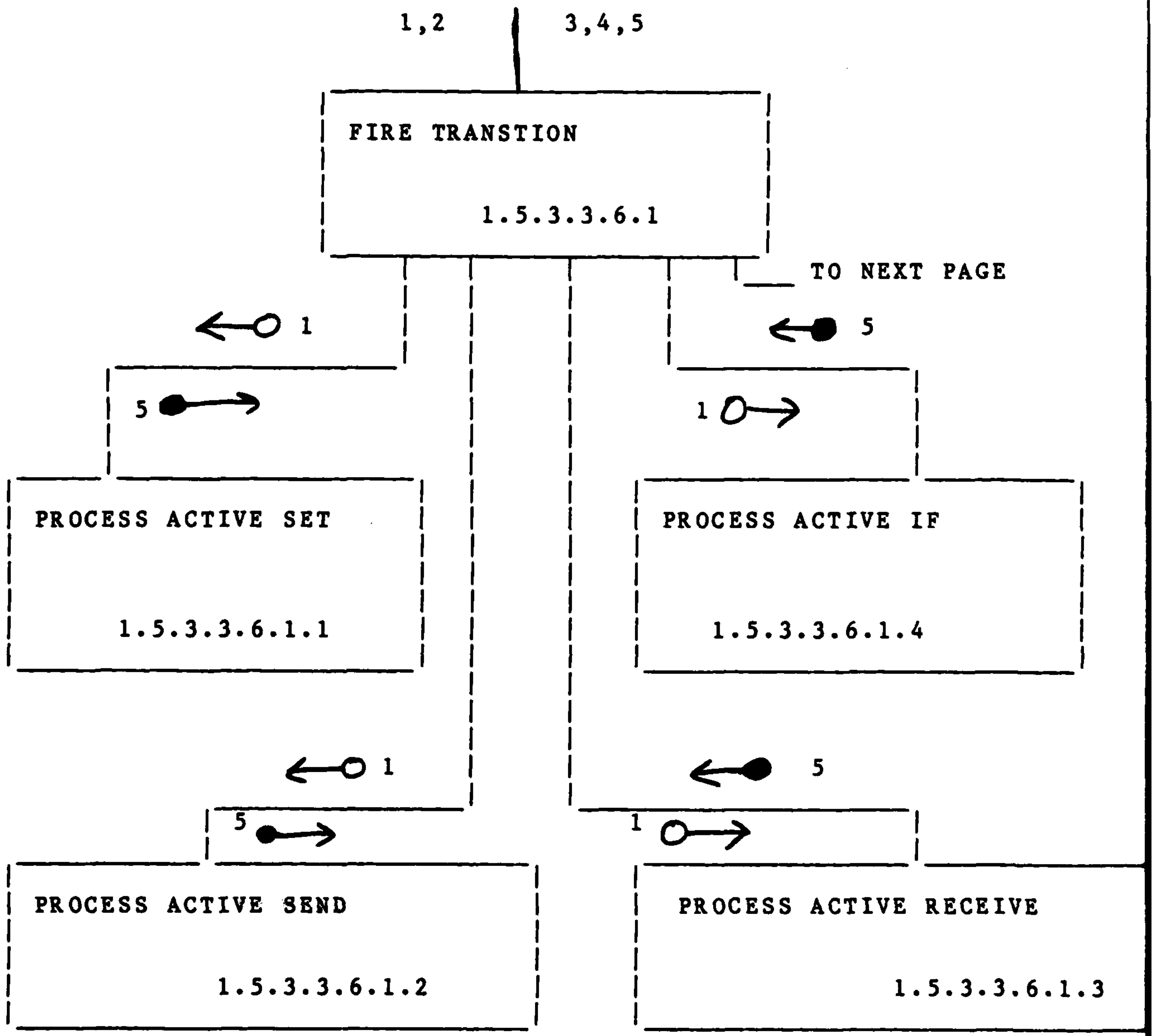
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. TRANSITION_THAT_FIRED
2. OUTPLACE
3. PLACE_PTR
4. ENABLED_TRANSITION

NODE:	TITLE:	NUMBER:
1.5.3.3.6.3	UPDATE EVENT LIST	

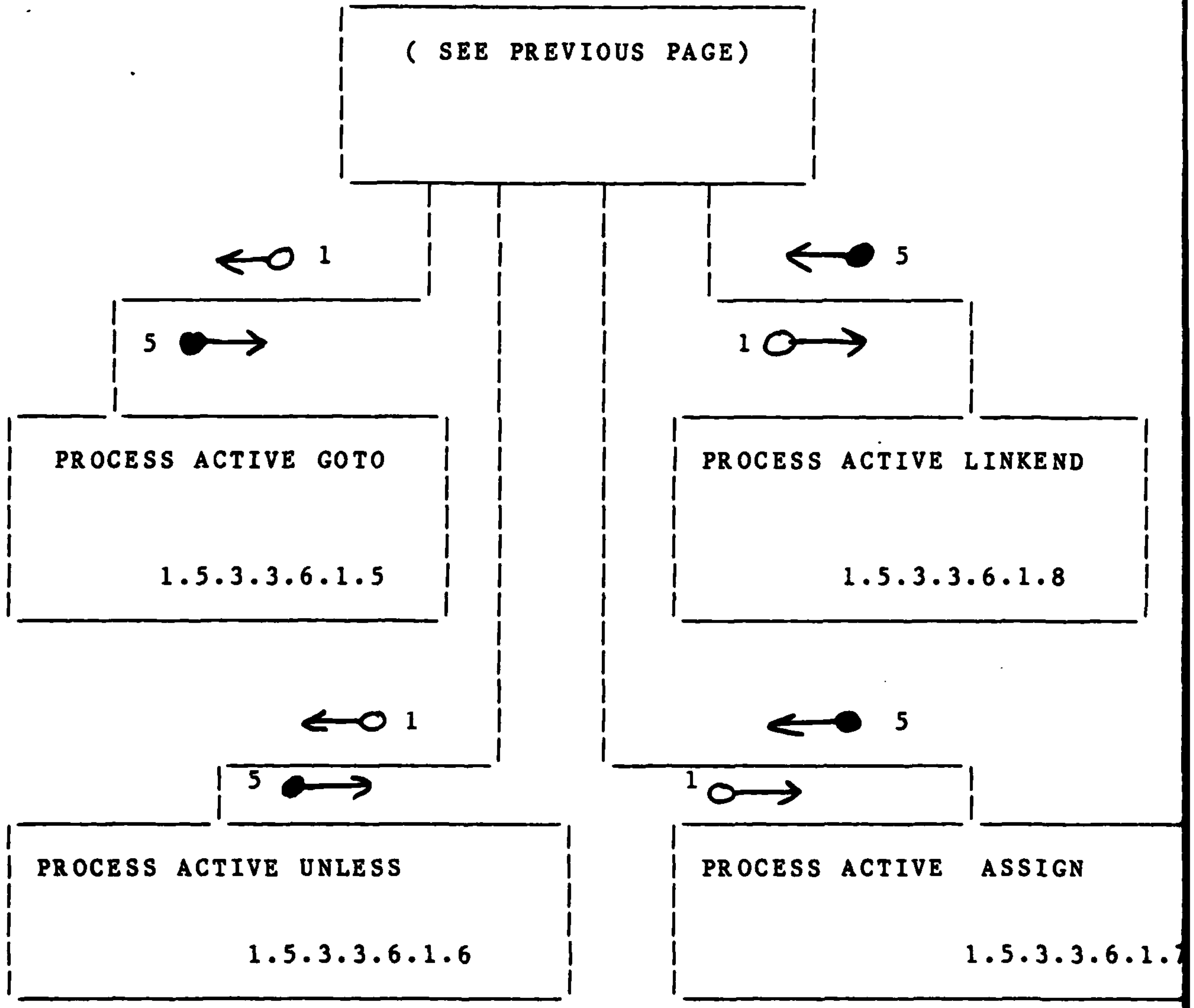
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. TRANSITION
2. PPML_ARRAY
3. OUTPLACE_1
4. OUTPLACE_2
5. ERROR_FLAG

NODE:	TITLE:	NUMBER:
1.5.3.3.6.1	FIRE TRANSITION (1 of 2)	

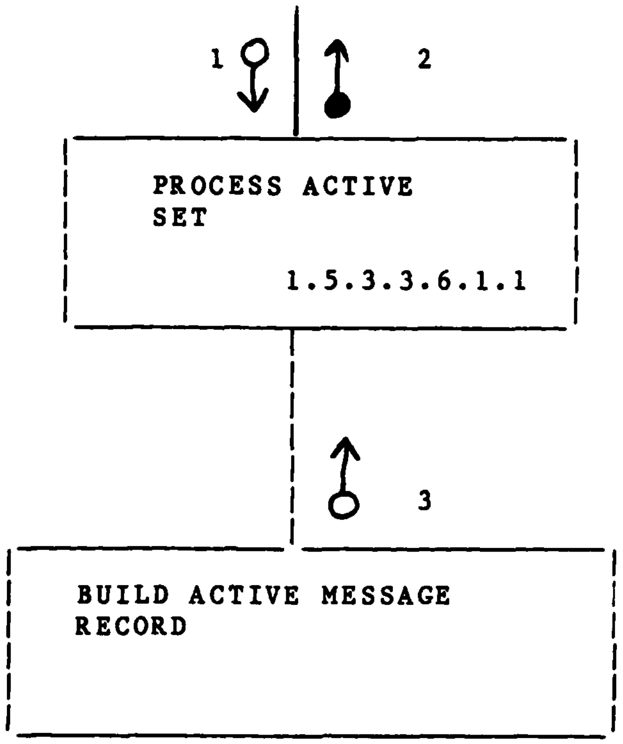
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. TRANSITION
2. PPML_ARRAY
3. OUTPLACE_1
4. OUTPLACE_2
5. ERROR_FLAG

NODE: 1.5.3.3.6.1	TITLE: FIRE TRANSITION (2 of 2)	NUMBER:

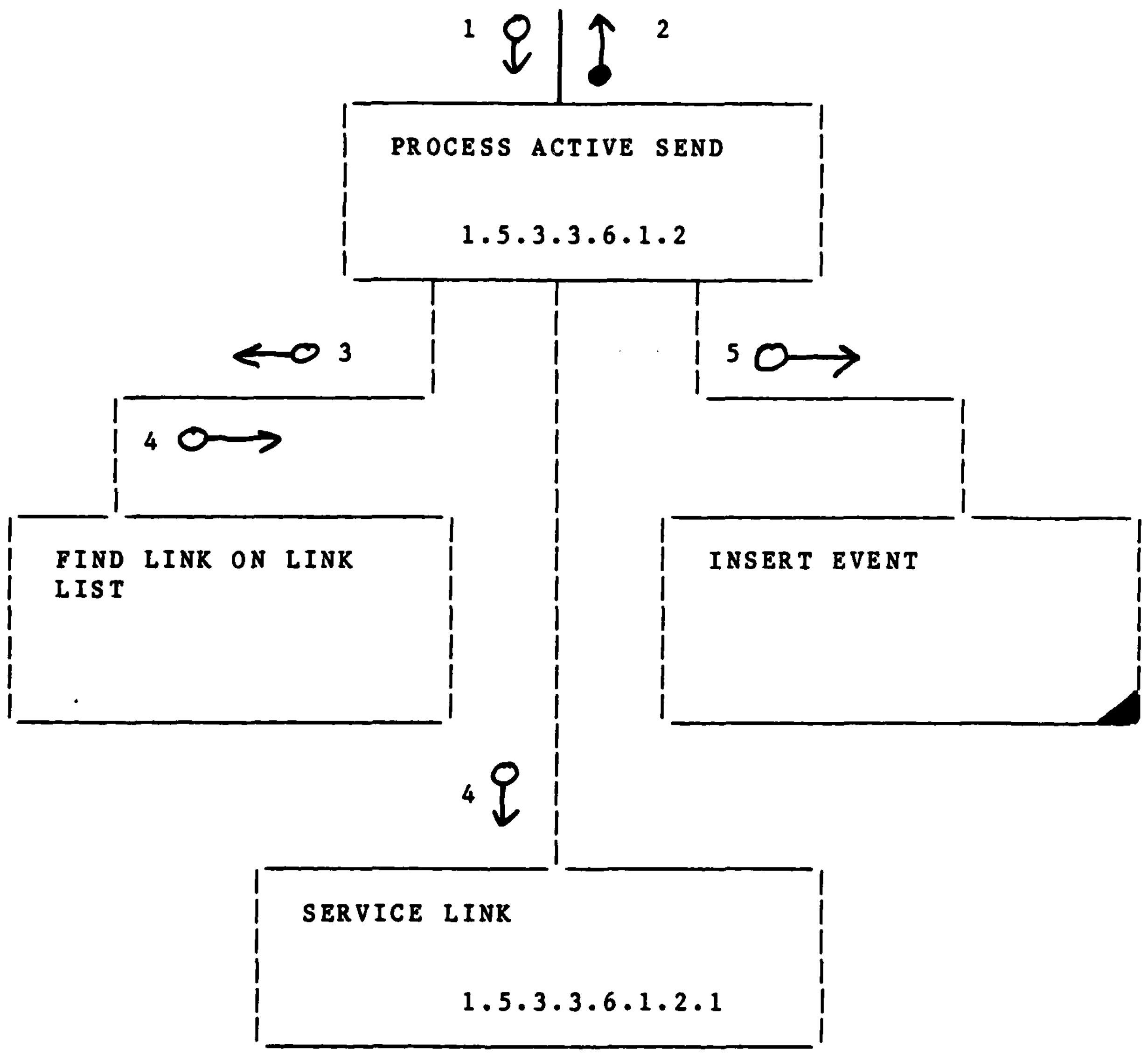
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. TRANSITION
2. ERROR_FLAG
3. ACTIVE_MESSAGE_POINTER

NODE: 1.5.	TITLE:	NUMBER:
3.3.6.1.1	PROCESS ACTIVE SET	

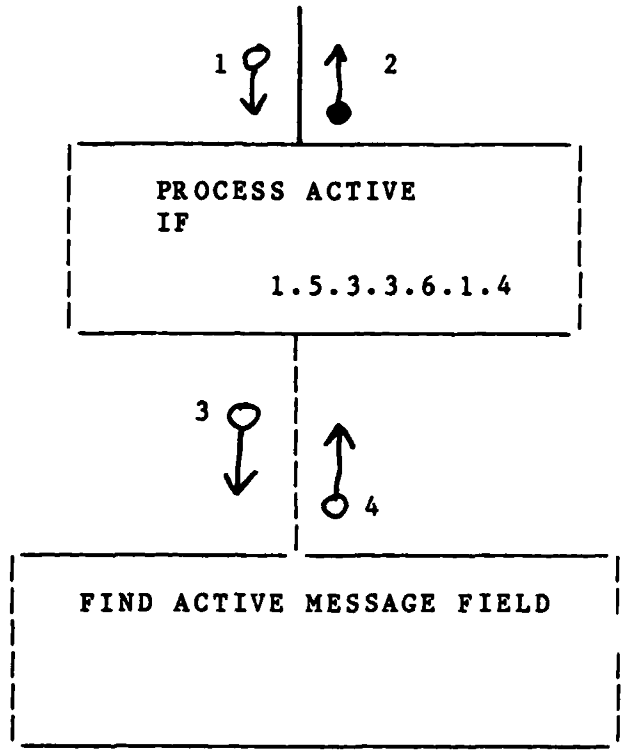
AUTHOR: K.R. MARTIN	PROJECT: AUTOMATED PROTOCOL VALIDATION SYSTEM	DATE: 28 NOV 83



1. TRANSITION
2. ERROR_FLAG
3. LINK_NAME
4. LINK_PTR
5. LINKEND_TRANSITION

NODE: 1.5. 3.3.6.1.2	TITLE: PROCESS ACTIVE SEND	NUMBER:

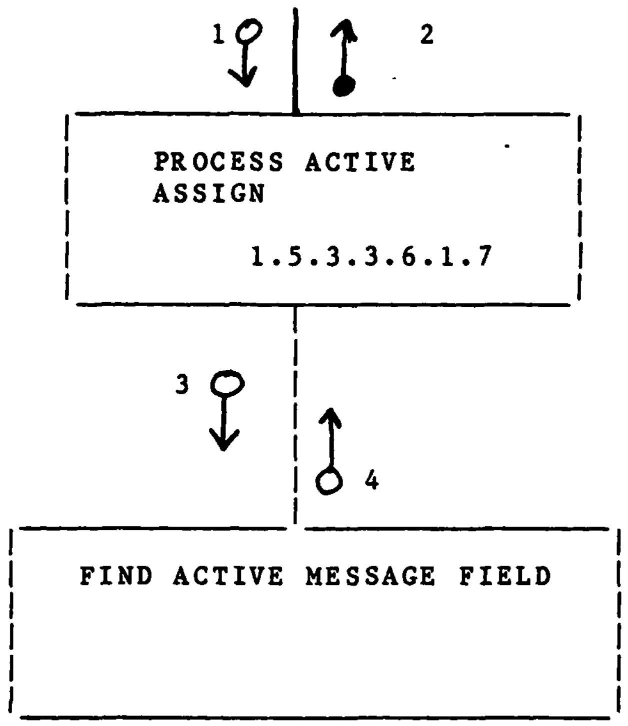
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. TRANSITION
2. ERROR_FLAG
3. MARKING
4. ACTIVE_MESSAGE_FIELD

NODE: 1.5	TITLE:	NUMBER:
.3.3.6.1.4	PROCESS ACTIVE IF	

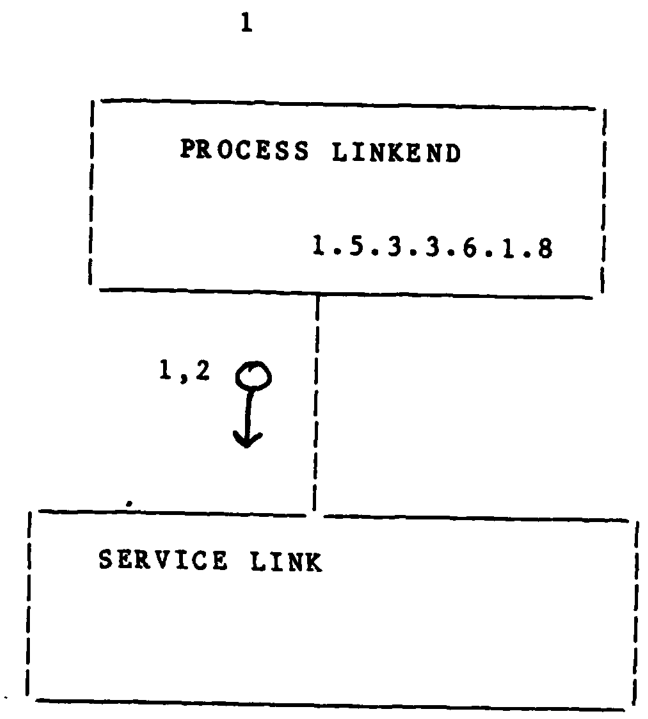
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



1. TRANSITION
2. ERROR_FLAG
3. MARKING
4. ACTIVE_MESSAGE_FIELD

NODE: 1.5.	TITLE:	NUMBER:
3.3.6.1.7	PROCESS ACTIVE ASSIGN	

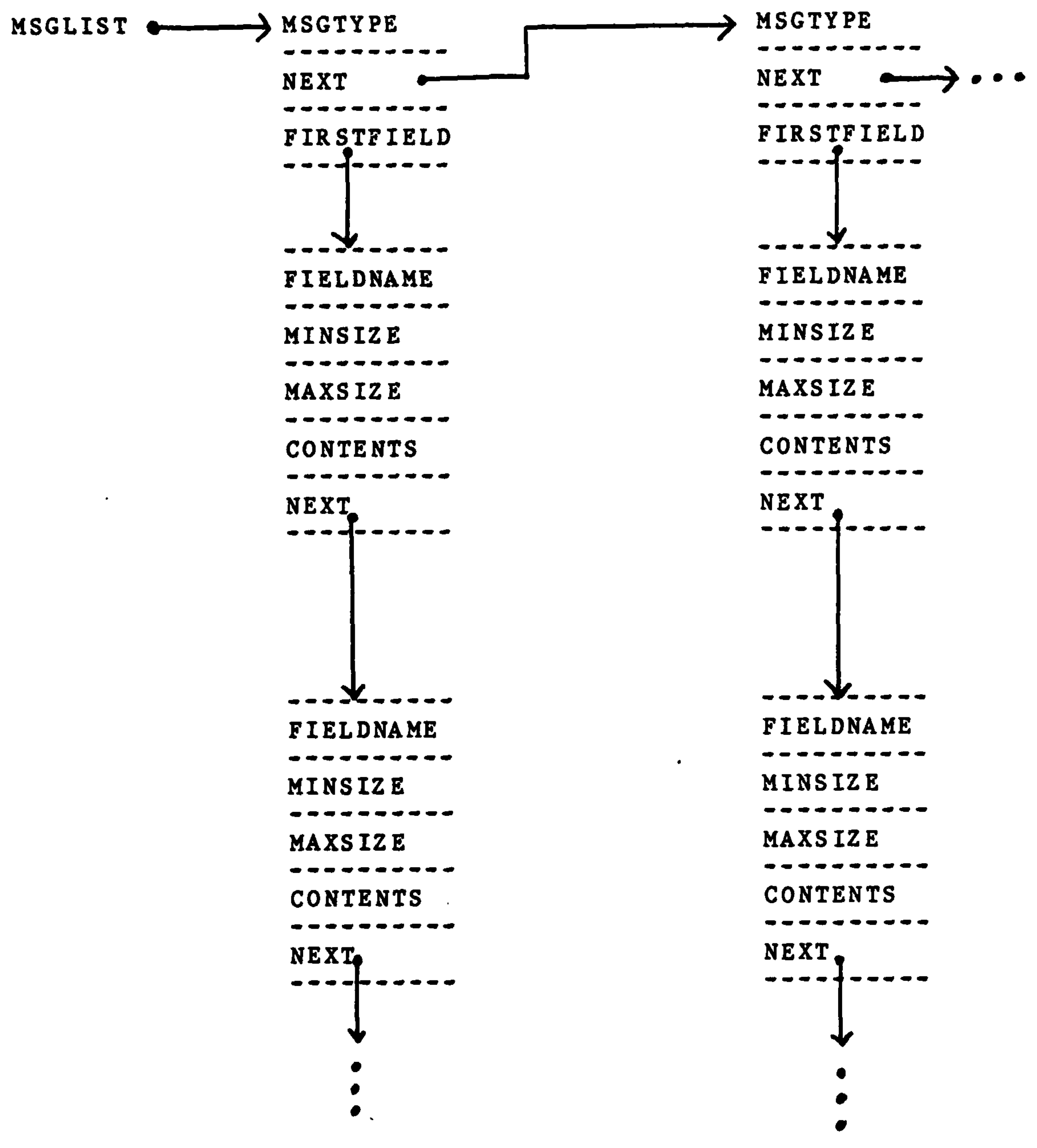
AUTHOR:	PROJECT: AUTOMATED	DATE:
K.R. MARTIN	PROTOCOL VALIDATION SYSTEM	28 NOV 83



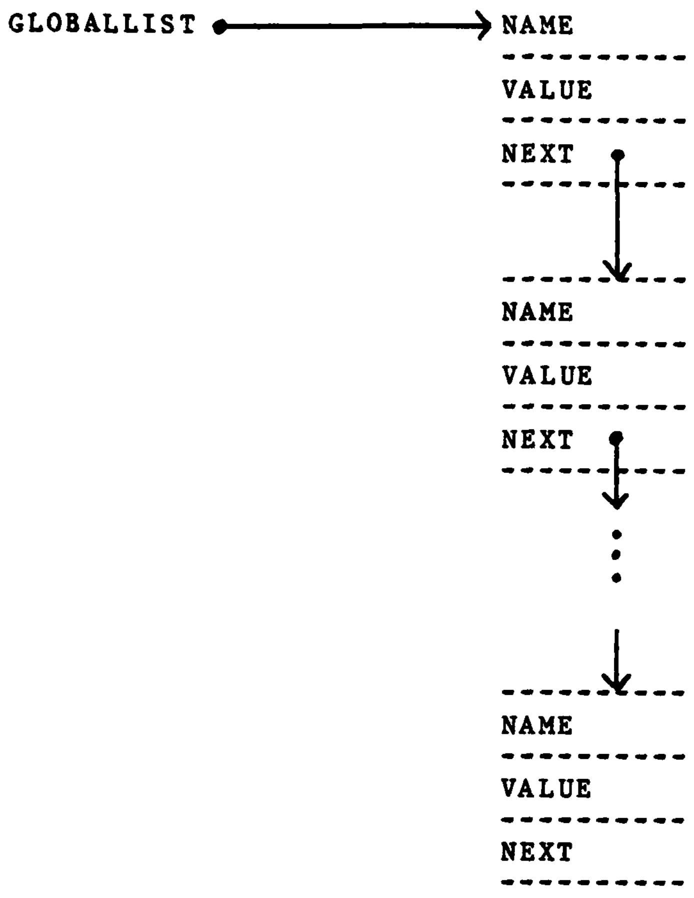
1. TRANSITION
2. LINK_NAME

NODE: 1.5. 3.3.6.1.8	TITLE: PROCESS ACTIVE LINK_END	NUMBER:

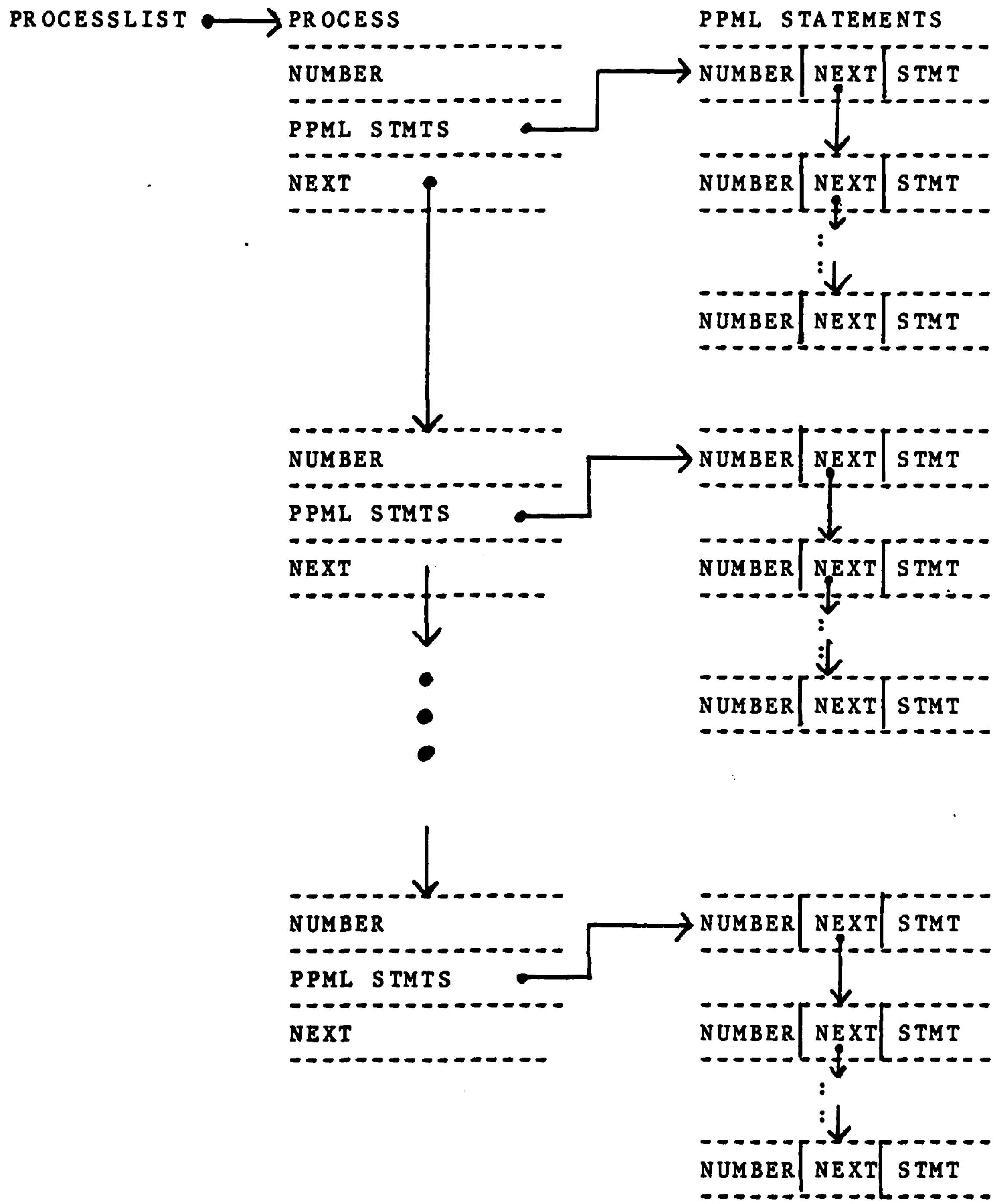
MESSAGE LIST



GLOBAL VARIABLE LIST



PROCESS LIST



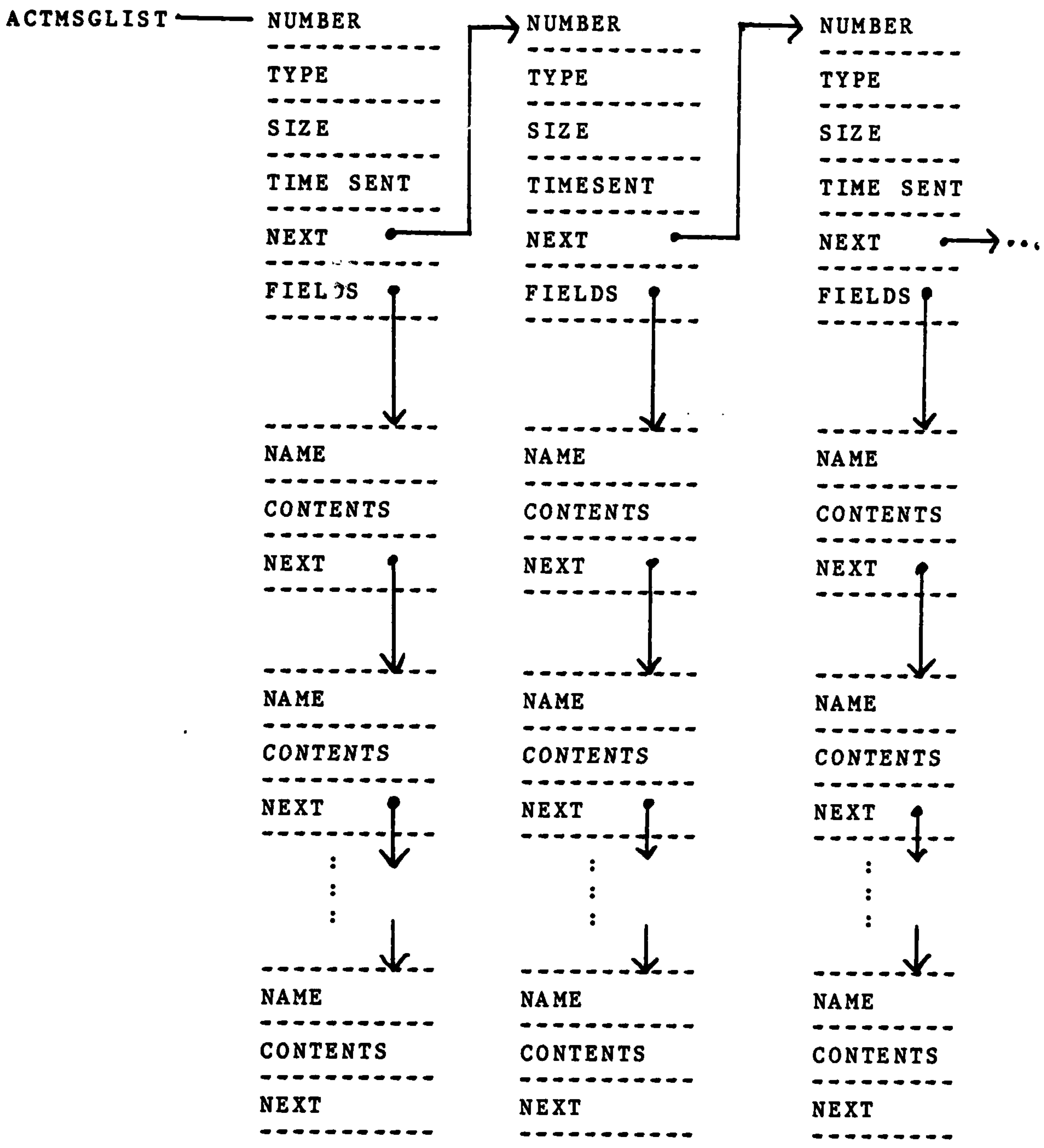
TRANSITION LIST

$$\begin{tikzcd}
\text{TRANSLIST} \arrow[r] & \boxed{\begin{array}{l}\text{TRANSITION}\\\text{DATA}\\\text{STRUCTURE}\end{array}} \arrow[r] & \boxed{\begin{array}{l}\text{TRANSITION}\\\text{DATA}\\\text{STRUCTURE}\end{array}} \arrow[r] & \boxed{\begin{array}{l}\text{TRANSITION}\\\text{DATA}\\\text{STRUCTURE}\end{array}} \arrow[r] & \cdots
\end{tikzcd}$$

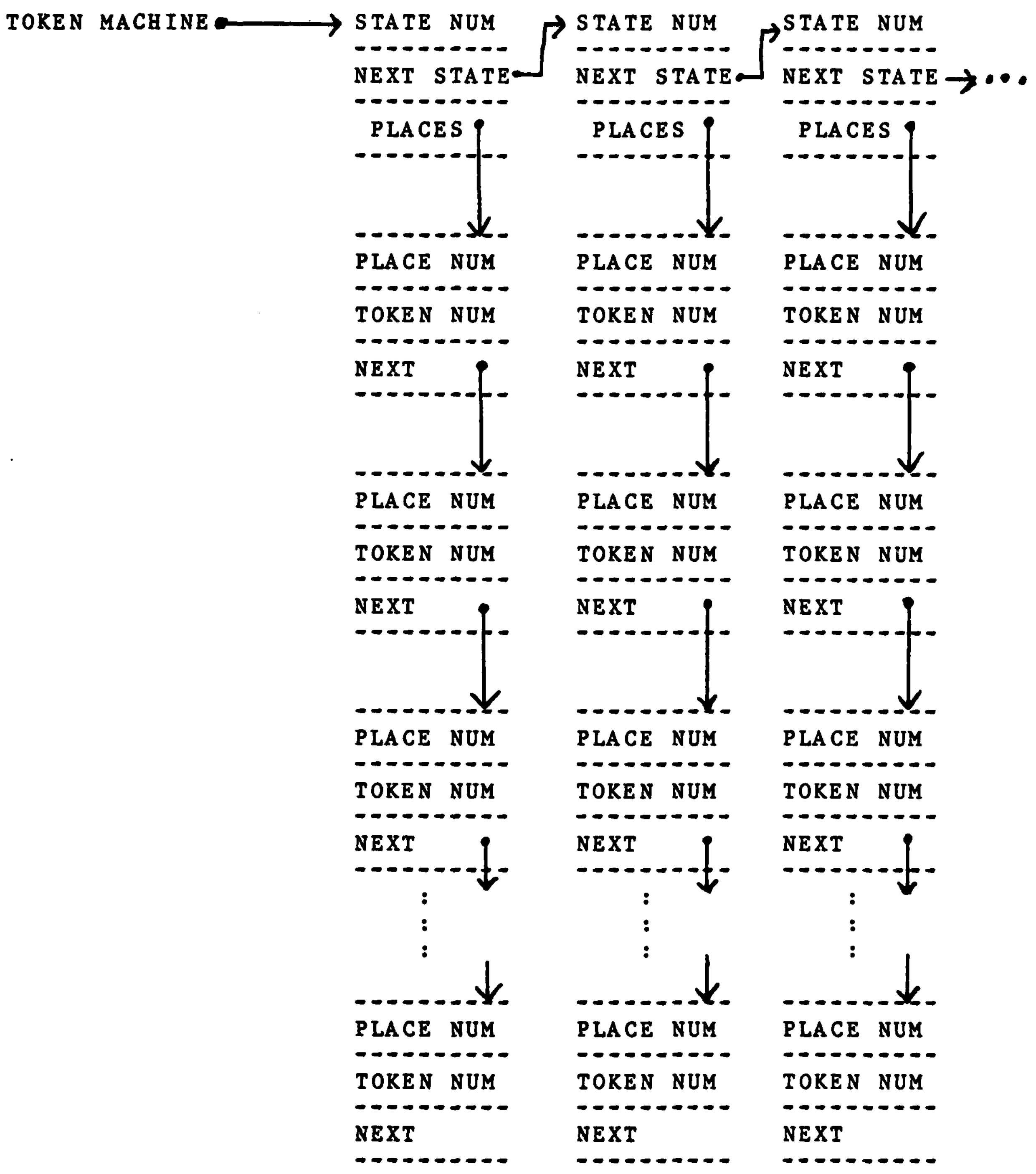
TRANSITION DATA STRUCTURE

NUMBER
FIRING TIME
STMT NUMBER
PROCESS NUMBER
STATEMENT TYPE
TRANSITION EXPRESSION
INPUT PLACE NUMBER 1 & POINTER
INPUT PLACE NUMBER 2 & POINTER
RESOLUTION PLACE NUMBER & POINTER
OUTPUT PLACE NUMBER 1 & POINTER
OUTPUT PLACE NUMBER 2 & POINTER
MESSAGE POINTER
TRANSITION PROCEDURE
NEXT TRANSITION ON LIST

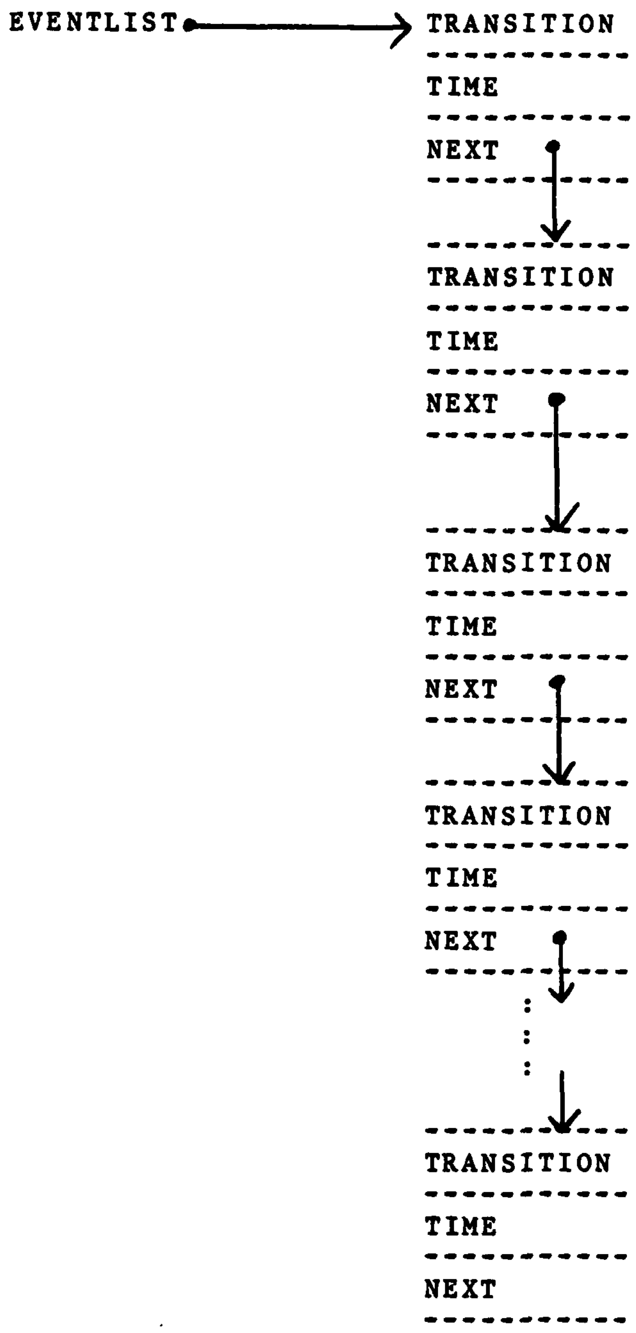
ACTIVE MESSAGE LIST



TOKEN MACHINE



EVENT LIST


EVENTLIST
TRANSITION
TIME
NEXT
TRANSITION
TIME
NEXT
TRANSITION
TIME
NEXT
TRANSITION
TIME
NEXT
TRANSITION
TIME
NEXT

APPENDIX D

TEST RESULTS

```
 Enter the number corresponding to the desired function.
1
  *** NEW MSG PROCESSING ***

 1- Enter the message type(10 or less characters)
M1
 ** Field Processing for Message Type-- M1
 Enter the Name of the Field.
F1
 Enter the minimun,maximum size of the field in bytes.
100
200
 More fields to input?(y or n)
n
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
4
 MESSAGE LIST
M1
FIELD NAME  MIN  MAX  CONTENTS
ERROR         0    0         0
F1          100  200         1
```

```
    *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
3
 *** Process Delete Message. ***
 ENTER THE NAME OF the MESSAGE to be DELETED.
M1
 MESSAGE FOUND
 MESSAGE TYPE -- M1
 ARE YOU SURE you WANT TO DELETE?(y or n)
y
 DELETED
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
4
 ** The Message list is empty **
```

```
     *** MASTER MESSAGE FORMAT MENU ***

  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
3
 *** Process Delete Message. ***
 ENTER THE NAME OF the MESSAGE to be DELETED.
M2
 MESSAGE TYPE M2         NOT FOUND
```

```
          ***  MASTER MENU  ***

      1 - ENTER MSG FORMAT MENU.
      2 - ENTER PPML FORMAT MENU.
      3 - ENTER EVALUATE PROTOCOL MENU.

 Enter the number corresponding to the desired function.
4
 ILLEGAL RESPONSE. TRY AGAIN.
```

```
     *** MASTER PPML FORMAT MENU ***

  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
9
 ILLEGAL RESPONSE. TRY AGAIN.
```

```
               *** MASTER GLOBAL VARIABLE MENU ***


   1 - ADD NEW VARIABLE
   2 - DELETE A VARIABLE
   3 - LIST VARIABLES
   4 - GO TO MASTER PPML MENU

  ENTER DESIRED FUNCTION NUMBER.
 5
               *** MASTER GLOBAL VARIABLE MENU ***
```

```
                *** MASTER GLOBAL VARIABLE MENU ***


  1 - ADD NEW VARIABLE
  2 - DELETE A VARIABLE
  3 - LIST VARIABLES
  4 - GO TO MASTER PPML MENU

 ENTER DESIRED FUNCTION NUMBER.
1
   *** Add Global Variable Processing ***
Enter the global Variable Name
CLOCK
Enter initial value of variable.
89
                *** MASTER GLOBAL VARIABLE MENU ***


  1 - ADD NEW VARIABLE
  2 - DELETE A VARIABLE
  3 - LIST VARIABLES
  4 - GO TO MASTER PPML MENU

 ENTER DESIRED FUNCTION NUMBER.
3
 *** GLOBAL VARIABLE LISTING ***

  Var Name              Value
CLOCK                          89
```

```
        *** MASTER PPML FORMAT MENU ***

     1 - Enter Global Variable
     2 - New PPML STATEMENTs
     3 - Change PPML Statement
     4 - Insert PPML statement
     5 - Delete PPML Statement
     6 - List PPML Statements
     7 - Input Number of Communication Links
     8 - Go To Master Menu
     * Enter Desired functioN number *
2
 * Enter Process number to work with. *
1
 ** New PPML statement processing **
 TO EXIT ENTER "Z" ONLY
|  LABEL | |PPML TYPE|| REMAINDER OF STATEMENT.....
---------------------------------------------------
START1      SET          { M1 }
            ASSIGN       { [F1] = [F1] + SEQ }
            SEND         { L1 }
WAITLP      RECEIVE      { L2 }
            UNLESS       { ACK1 } WAITLP
            IF           { [ERROR] = 1 } THEN WAITLP
            END
Z
```

```
        *** MASTER PPML FORMAT MENU ***

     1 - Enter Global Variable
     2 - New PPML STATEMENTs
     3 - Change PPML Statement
     4 - Insert PPML statement
     5 - Delete PPML Statement
     6 - List PPML Statements
     7 - Input Number of Communication Links
     8 - Go To Master Menu
     * Enter Desired functioN number *
2
 * Enter Process number to work with. *
2
 ** New PPML statement processing **
 TO EXIT ENTER "Z" ONLY
|  LABEL |  |PPML TYPE|| REMAINDER OF STATEMENT.....
-----------------------------------------------------
START2      RECEIVE     { L1 }
            UNLESS      { M1 } START2
            IF          { [ERROR] = 1 } THEN START2
            ASSIGN      { [TIMERCVD] = CLOCK }
            SET         { ACK1 }
            SEND        { L2 }
            GOTO        START2
            END
Z
```

```
** Processing ***.  statement     1 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          1
** Processing ***.  statement     3 PROCESS          1
** Processing ***.  statement     4 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS          1
** Processing ***.  statement     6 PROCESS          1
** Processing ***.  statement     7 PROCESS          1
** Processing ***.  statement     1 PROCESS          2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          2
** Processing ***.  statement     3 PROCESS          2
** Processing ***.  statement     4 PROCESS          2
** Processing ***.  statement     5 PROCESS          2
** Processing ***.  statement     6 PROCESS          2
** Processing ***.  statement     7 PROCESS          2
** Processing ***.  statement     8 PROCESS          2
  E-NET built
  * Enter Desired functioN number *
3
 * Enter Process number to work with. *
2
 ** Change PPML statement processing **
 Enter the number of the statement to be changed
 in process number         2
2
           UNLESS      { M1 } START2
 Enter new PPML statement:
           SET         { X1 }
    *** MASTER PPML FORMAT MENU ***


  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
6
 * Enter Process number to work with. *
2
 ** List PPML Processing **
      ** PPML STATEMENTS FOR PROCESS          2


  STATEMENT #              STATEMENT
         1START2      RECEIVE     { L1 }
         2            SET         { X1 }
         3            IF          { [ERROR] = 1 } THEN START2
         4            ASSIGN      { [TIMERCVD] = CLOCK }
         5            SET         { ACK1 }
         6            SEND        { L2 }
         7            GOTO        START2
         8            END
  * Enter Desired functioN number *
5
 * Enter Process number to work with. *
1
 ** Delete PPML processing **
Enter number of statement to be deleted.
2
Are you sure you want to delete.(y or n)
y
     *** MASTER PPML FORMAT MENU ***


  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
6
 * Enter Process number to work with. *
1
 ** List PPML Processing **
       ** PPML STATEMENTS FOR PROCESS          1


  STATEMENT #               STATEMENT
         1START1     SET        { M1 }
         2           SEND       { L1 }
         3WAITLP     RECEIVE    { L2 }
         4           UNLESS     { ACK1 } WAITLP
         5           IF         { [ERROR] = 1 } THEN WAITLP
         6           END
  * Enter Desired functioN number *
4
 * Enter Process number to work with. *
1
Enter statement number to be inserted .
2
Enter new statement number    2
| LABEL  | |PPML TYPE||REMAINDER OF STATEMENT...
           ASSIGN     { [F1] = [F1] + SEQ }
Last number          2
Last number          3
Last number          4
Last number          5
Last number          6
     *** MASTER PPML FORMAT MENU ***


  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
6
 * Enter Process number to work with. *
1
 ** List PPML Processing **
       ** PPML STATEMENTS FOR PROCESS          1


  STATEMENT #             STATEMENT
         1START1     SET         { M1 }
         2           ASSIGN      { [F1] = [F1] + SEQ }
         3           SEND        { L1 }
         4WAITLP     RECEIVE     { L2 }
         5           UNLESS      { ACK1 } WAITLP
         6           IF          { [ERROR] = 1 } THEN WAITLP
         7           END
  * Enter Desired functioN number *
3
 * Enter Process number to work with. *
1
 ** Change PPML statement processing **
 Enter the number of the statement to be changed
 in process number           1
12
Error in Findppml. Stnum   12 process          1
```

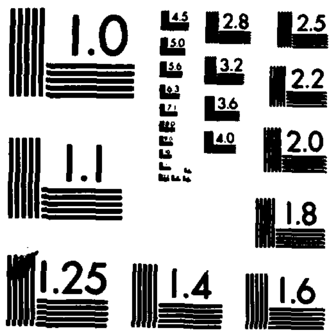
```
** Processing ***.  statement     1 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          1
** Processing ***.  statement     3 PROCESS          1
** Processing ***.  statement     4 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS          1
 ERROR---Could not find msg type **  ACK1      ** in UNLESS
** Processing ***.  statement     6 PROCESS          1
** Processing ***.  statement     7 PROCESS          1
** Processing ***.  statement     1 PROCESS          2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          2
** Processing ***.  statement     3 PROCESS          2
** Processing ***.  statement     4 PROCESS          2
** Processing ***.  statement     5 PROCESS          2
ERROR *** COULD NOT FIND MSG ACK1
** Processing ***.  statement     6 PROCESS          2
** Processing ***.  statement     7 PROCESS          2
** Processing ***.  statement     8 PROCESS          2
  E-NET built
```

```
** Processing ***.  statement     1 PROCESS            1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS            1
** Processing ***.  statement     3 PROCESS            1
 ***ERROR ***.
 *** ILLEGAL STATEMENT TYPE ***
*** ERROR ***
  PROCESS          1
  STATEMENT          3
** Processing ***.  statement     4 PROCESS            1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS            1
** Processing ***.  statement     6 PROCESS            1
** Processing ***.  statement     7 PROCESS            1
** Processing ***.  statement     1 PROCESS            2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS            2
** Processing ***.  statement     3 PROCESS            2
** Processing ***.  statement     4 PROCESS            2
** Processing ***.  statement     5 PROCESS            2
** Processing ***.  statement     6 PROCESS            2
** Processing ***.  statement     7 PROCESS            2
** Processing ***.  statement     8 PROCESS            2
  E-NET built
segmentation fault (core dumped)
```

```
** Processing ***.  statement     1 PROCESS            1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS            1
** Processing ***.  statement     3 PROCESS            1
 ** COULD NOT FIND LINK NAME ** LX
** Processing ***.  statement     4 PROCESS            1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS            1
** Processing ***.  statement     6 PROCESS            1
** Processing ***.  statement     7 PROCESS            1
** Processing ***.  statement     1 PROCESS            2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS            2
** Processing ***.  statement     3 PROCESS            2
** Processing ***.  statement     4 PROCESS            2
** Processing ***.  statement     5 PROCESS            2
** Processing ***.  statement     6 PROCESS            2
** Processing ***.  statement     7 PROCESS            2
** Processing ***.  statement     8 PROCESS            2
  E-NET built
 Auto execution processing
 Enter number of events to process.
30
 RCV Processing here **
 Set Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 Linkend Processing here **
 Do you want msg number ,          1
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Unless Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 RCV Processing here **
 If  Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 Set Processing here **
 Unless Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 RCV Processing here **
 Linkend Processing here **
 Do you want msg number ,          2
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Goto Processing here **
 Unless Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 RCV Processing here **
 If  Processing here **
 Enter Option Number
3
 **  ACTMSGLIST  **
 NUMBER | TYPE | SIZE | TIME SENT | TIME RCVD | FLD NAME | CONTENTS
 ---------------------------------------------------------------
      2 | ACK1 |   20 |      9.00 |     12.00
                                              ERROR              0
                                              F1                 0
      1 | M1   |  200 |      1.00 |      8.00
                                              ERROR            234
                                              F1                 0
      0 | TOKEN      |     1 |      0.00 |      0.00
 Enter Option Number
2
 ** SHOW EVENTLIST PROCESSING HERE **
 TIME | TRANSNUM | PROC | STMT
______________________________________
 15.00          16       2       2
```

```
                 **** E-NET STRUCTURE ****

TRANS | PROC |  ST  |INPL1 |INPL2 |RESPL |OUTPL1|OUTPL2|
==========================================================
    0|     0|     0|     0|     0|      |     0|     0
    1|   101|     0|     2|      |      |     1|
    2|   101|     0|     1|      |      |     3|
    3|   101|     0|     3|      |      |     4|
    4|   102|     0|     6|      |      |     5|
    5|   102|     0|     5|      |      |     7|
    6|   102|     0|     7|      |      |     8|
    7|     1|     1|    10|      |      |     9|
    8|     1|     2|     9|      |      |    11|
    9|     1|     3|    11|      |      |    20|     2
   10|     1|     4|    20|     8|      |    14|
   11|     1|     5|    14|      |     1|    15|    20
   12|     0|     0|    16|    12|     2|    17|
   13|     1|     6|    15|      |     3|    18|    20
   14|     0|     0|    19|    17|     4|    20|
   15|     2|     1|    34|     4|      |    21|
   16|     2|     2|    21|      |     5|    23|    34
   17|     0|     0|    24|    22|     6|    25|
   18|     2|     3|    23|      |     7|    26|    34
   19|     0|     0|    27|    25|     8|    28|
   20|     2|     4|    26|      |      |    29|
   21|     2|     5|    29|      |      |    30|
   22|     2|     6|    30|      |      |    31|     6
   23|     2|     7|    31|      |      |      |    34
   24|     0|     0|    33|    28|     9|    34|
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
5
 RCV Processing here **
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
 Enter Option Number
1
 ** TOKEN MACHINE STATE NUMBER **       1
 PLACE NUMBER        0  TOKEN       0
 ** TOKEN MACHINE STATE NUMBER **       2
 PLACE NUMBER       34  TOKEN       0
 PLACE NUMBER       10  TOKEN       0
 ** TOKEN MACHINE STATE NUMBER **       3
 PLACE NUMBER       21  TOKEN       0
 PLACE NUMBER       10  TOKEN       0
 ** TOKEN MACHINE STATE NUMBER **       4
 PLACE NUMBER       21  TOKEN       0
 PLACE NUMBER        9  TOKEN       1
 Enter Option Number
4
 ** SHOWTOKENPLACEMENT PROCESSING HERE **
 PLACE NUMBER | TOKEN NUMBER
        11    |      1
        34    |      0
```

AD-A138 160   AN AUTOMATED COMPUTER COMMUNICATION NETWORK PROTOCOL   3/3
VERIFICATION SYSTEM(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
UNCLASSIFIED   K R MARTIN DEC 83 AFIT/GCS/EE/83D-12   F/G 9/2   NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX E

HDLC protocol evaluation

HDLC specification in PPML

1) The first information to be gathered in the validation effort is information on the types of messages used by the system and their formats. The HDLC formats that will be used in this development are described by Tannenbaum in his "Computer Networks" textbook.

The format of the messages is as follows:

Bits	8	8	8	>=0	16	8
	start	Address	Control	Data	Cksum	

The start and end fields are special patterns that delimit the message.

The Address field is used primarilly for identification of terminals on multidrop lines and will not be used in our analysis.

The Control field is used for sequence numbers, acknowledgements, and other purposes. They will be described shortly.

The Data field may contain arbitrary information. It may be arbitrarily long.

The Cksum is a variation of the CRC which uses the CRC-CCITT as the generating polynomial.

There are three kinds of frames: Information, Supervisory, and Unnumbered. The control field of each of these is different and we can use this to model the messages. The formats that will represent these messages are as follows.

Message Type	Fields
INFO	SEQ NEXT DATA
TYPE0	NEXT
TYPE1	NEXT
TYPE2	NEXT
TYPE3	NEXT
CTRLACK	DATA

The INFO frame contains a sequence number and the number of the next frame expected from the other party as well as data.

The TYPE0 through TYPE3 frames are Supervisory frames. The TYPE0 is an acknowledgement frame used when no information frame is available to piggyback the acknowledgement. It acknowledges all frames up to but not including the frame with sequence number equal to the NEXT field.

The TYPE1 is a negative acknowledegement. The NIEX field indicates the sequence number of the first frame received incorrectly. The sender must then retransmit all outstanding frames from that sequence number on.

The TYPE2 is a RECEIVER NOT READY. It acknowledges all frames up to the NEXT field but requests the sender to stop sending until he receives a TYPE0 message.

The TYPE3 is a selective reject message. It calls for retransmission of only the frame indicated.

The CTRLACK is used to acknowledge control frames. Since we are not going to model this portion of the protocol we do not need this message type.

The protocol uses a sliding window , with a 3 bit sequence number. Up to seven unacknowledged frames may be outstanding at any instant. The SEQ field of the INFO message contains the sequence number. The NEXT field of the messages is a piggybcked acknowledgement.

Now that we have decided on the message format we can work on the PPML statements needed to represent the protocol. The basic activity of any protocol is to receive a message and act according to the type of message received. The actions of the HDLC are defined as in the previous section. So, it is straightforward process to transform the specification into the PPML description. We will model a two party communication system where the slystem has already been initialized and is now in a data transfer mode. One party is the SENDING process and the other is the RECEIVING process. We will also assume that the receiver has no INFO frames to piggyback acknowledgements and will acknowledge all frames received.

```
PROCESS 1 - SENDER

GLOBALS     -     SEQNUM=1
                  BOTWIN=1
                  TOPWIN=7
                  SEQNUMSV=0

PPML

SENDMSG    IF         { SEQNUM <> BOTWIN-1 } THEN CKBOT
           GOTO       WAIT
CKBOT      SET        { INFO }
           ASSIGN     { [SEQ] = SEQNUM }
           ASSIGN     { SEQNUM = SEQNUM + 1}
           IF         { SEQNUM = 8 } THEN ADJUST
SEND1      SEND       { L1 }
           RECEIVE    { L2 }
           UNLESS     { BASETOKEN } PROCESSRCV
           GOTO       SENDMSG
PROCESSRCV UNLESS     { TYPE0 } CKT1
           ASSIGN     {BOTWIN = [NEXT]}
           ASSIGN     { TOPWIN = BOTWIN + 6 }
           GOTO       SENDMSG
CKT1       UNLESS     { TYPE1 } CKT2
           ASSIGN     { BOTWIN = [NEXT] }
           ASSIGN     { TOPWIN = BOTWIN + 6 }
           GOTO       SENDMSG
CKT2       UNLESS     { TYPE2 } CKT3
           ASSIGN     { BOTWIN = [NEXT] }
           ASSIGN     { TOPWIN = BOTWIN + 6 }
           GOTO       WAIT
CKT3       UNLESS     { TYPE3 } SENDMSG
           ASSIGN     { SEQNUMSV = [NEXT] }
           SET        { INFO }
           ASSIGN     { [SEQ] = SEQNUMSV }
           SEND       { L1 }
           GOTO       SENDMSG
WAIT       RECEIVE    { L2 }
           UNLESS     { BASETOKEN } PROCESSRCV
           GOTO       WAIT
ADJUST     ASSIGN     { SEQNUM = 0 }
           GOTO       SEND1
           END
```

PROCESS 2 - RECEIVER

GLOBALS - NEXT
          CLOCK

PPML for PROCESS 2

```
RCV        RECEIVE     { L1 }
           UNLESS      { BASETOKEN } PROCESSMSG
           GOTO        RCV
PROCESSMSG IF          { [ERROR] = 1 } THEN PROCESSERR
           UNLESS      { INFO } PROCESSCTR
           ASSIGN      { [TIMERCVD] = CLOCK }
           ASSIGN      { NEXT = [SEQ] + 1 }
           IF          { NEXT < 8   } THEN CONTINUE
           ASSIGN      { NEXT = 0 }
CONTINUE   SET         { TYPE0 }
           ASSIGN      { [NEXT]=NEXT }
           SEND        { L2 }
           GOTO        RCV
PROCESSERR SET         { TYPE1 }
           ASSIGN      { [NEXT] = NEXT }
           SEND        { L2 }
           GOTO        RCV
           END
```

Now that the message formats and PPML descriptions have been decided upon, they can be entered into the system. The order of entry would be to enter the message formats, then the global variables, then the number of communication links needed(i.e 2), and then the PPML descriptions of the processes. Once the entry phase has been completed, the evaluation phase begins. The system will request information on the communication links. The number of messages allowed on the links at one time is equal to the maximum number of outstanding messages, which is 7. The transmission times are dependent on the baud rate of the links and the distance between the two communicating processes.

Once the communication link data has been entered, the system will build the E-Net and report any syntax errors detected. After the E-Net has been successfully built, the user will be prompted as to the mode of simulation desired, the auto or manual mode. The auto mode will ask for the number of events to process and then begin execution. It will automatically insert errors and lose messages as specified by the link descriptions.

Once the simulation has been completed, the token machine and active message list can be examined to complete the analysis.

As with any simulation, this process represents the action of the described system under the parameters specified. There is no guarantee that all possible sequences have been simulated. The user can gain more control over the processing by operating in the manual mode. The manual mode is most useful when desiring to examine the effects of certain sequences of message transmission and error situations.

Appendix F

Stop and Wait Protocol Session

```
VVSYS

 Enter the number of communicating processes.
 The number of processes is     2
      ***  MASTER MENU  ***


  1 - ENTER MSG FORMAT MENU.
  2 - ENTER PPML FORMAT MENU.
  3 - ENTER EVALUATE PROTOCOL MENU.


 Enter the number corresponding to the desired function.
1
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
1
  *** NEW MSG PROCESSING ***

 1- Enter the message type(10 or less characters)
M1
 ** Field Processing for Message Type-- M1
 Enter the Name of the Field.
F1
 Enter the minimun,maximum size of the field in bytes.
1000
2000
 More fields to input?(y or n)
n
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
1
  *** NEW MSG PROCESSING ***

 1- Enter the message type(10 or less characters)
M1
 ** Field Processing for Message Type-- M1
 Enter the Name of the Field.
F1
 Enter the minimun,maximum size of the field in bytes.
1000
2000
 More fields to input?(y or n)
n
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
1
  *** NEW MSG PROCESSING ***

 1- Enter the message type(10 or less characters)
ACK
 ** Field Processing for Message Type-- ACK
 Enter the Name of the Field.
F1
 Enter the minimun,maximum size of the field in bytes.
20
20
 More fields to input?(y or n)
n
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
5
 Going to Master menu.
      ***  MASTER MENU  ***


  1 - ENTER MSG FORMAT MENU.
  2 - ENTER PPML FORMAT MENU.
  3 - ENTER EVALUATE PROTOCOL MENU.


 Enter the number corresponding to the desired function.
2
     *** MASTER PPML FORMAT MENU ***
```

```
    1 - Enter Global Variable
    2 - New PPML STATEMENTs
    3 - Change PPML Statement
    4 - Insert PPML statement
    5 - Delete PPML Statement
    6 - List PPML Statements
    7 - Input Number of Communication Links
    8 - Go To Master Menu
    * Enter Desired functioN number *
1
           *** MASTER GLOBAL VARIABLE MENU ***


    1 - ADD NEW VARIABLE
    2 - DELETE A VARIABLE
    3 - LIST VARIABLES
    4 - GO TO MASTER PPML MENU

 ENTER DESIRED FUNCTION NUMBER.
1
   *** Add Global Variable Processing ***
Enter the global Variable Name
CLOCK
Enter initial value of variable.
100
           *** MASTER GLOBAL VARIABLE MENU ***


    1 - ADD NEW VARIABLE
    2 - DELETE A VARIABLE
    3 - LIST VARIABLES
    4 - GO TO MASTER PPML MENU

 ENTER DESIRED FUNCTION NUMBER.
1
   *** Add Global Variable Processing ***
Enter the global Variable Name
WAITCTR
Enter initial value of variable.
20
           *** MASTER GLOBAL VARIABLE MENU ***


    1 - ADD NEW VARIABLE
    2 - DELETE A VARIABLE
    3 - LIST VARIABLES
    4 - GO TO MASTER PPML MENU

 ENTER DESIRED FUNCTION NUMBER.
4
Going to PPML Menu
     *** MASTER PPML FORMAT MENU ***
   1 - Enter Global Variable
   2 - New PPML STATEMENTs
   3 - Change PPML Statement
   4 - Insert PPML statement
   5 - Delete PPML Statement
   6 - List PPML Statements
   7 - Input Number of Communication Links
   8 - Go To Master Menu
   * Enter Desired functioN number *
7
 Enter the number of communication links.
2
     *** MASTER PPML FORMAT MENU ***


   1 - Enter Global Variable
   2 - New PPML STATEMENTs
   3 - Change PPML Statement
   4 - Insert PPML statement
   5 - Delete PPML Statement
   6 - List PPML Statements
   7 - Input Number of Communication Links
   8 - Go To Master Menu
   * Enter Desired functioN number *
2
 * Enter Process number to work with. *
1
 ** New PPML statement processing **
 TO EXIT ENTER "Z" ONLY
|  LABEL  |  |PPML TYPE|| REMAINDER OF STATEMENT.....
----------------------------------------------------
START1     SET        { M1 }
           SEND       { L1 }
           ASSIGN     { WAITCTR = 20 }
WAITLP     RECEIVE    { L2 }
           UNLESS     { ACK } WAITLP2
           GOTO       SENDM2
WAITLP2    ASSIGN     { WAITCTR = WAITCTR - 1}
           IF         { WAITCTR = 0 } THEN START1
           GOTO       WAITLP
SENDM2     SET        { M2 }
           SEND       { L1 }
           ASSIGN     { WAITCTR = 20 }
WAITLP3    RECEIVE    { L2 }
           UNLESS     { ACK } WAITLP4
           GOTO       START1
WAITLP4    ASSIGN     { WAITCTR = WAITCTR - 1 }
           IF         { WAITCTR = 0 } THEN SENDM2
           GOTO       WAITLP3
           END
Z
     *** MASTER PPML FORMAT MENU ***
```

```
  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
2
 * Enter Process number to work with. *
2
 ** New PPML statement processing **
 TO EXIT ENTER "Z" ONLY
|  LABEL | |PPML TYPE|| REMAINDER OF STATEMENT.....
----------------------------------------------------
START2      RECEIVE    { L1 }
            UNLESS     { M1 } WAITM1
            SET        { ACK }
            SEND       { L2 }
            GOTO       WAITM2
WAITM1      UNLESS     { M2 } START2
            SET        { ACK }
            SEND       { L2 }
            GOTO       START1
WAITM2      RECEIVE    { L1 }
            UNLESS     { M2 } WAIT3
            SET        { ACK }
            SEND       { L2 }
            GOTO       START2
WAIT3       UNLESS     { M1 } WAITM2
            SET        { ACK }
            SEND       { L2 }
            GOTO       WAITM2
            END
Z
     *** MASTER PPML FORMAT MENU ***


  1 - Enter Global Variable
  2 - New PPML STATEMENTs
  3 - Change PPML Statement
  4 - Insert PPML statement
  5 - Delete PPML Statement
  6 - List PPML Statements
  7 - Input Number of Communication Links
  8 - Go To Master Menu
  * Enter Desired functioN number *
8
 Going to Master Menu.
      ***  MASTER MENU  ***
  1 - ENTER MSG FORMAT MENU.
  2 - ENTER PPML FORMAT MENU.
  3 - ENTER EVALUATE PROTOCOL MENU.


 Enter the number corresponding to the desired function.
3
  ** LINK PROCESSING **
Building Link number          1
 ENTER LINK NAME
L1
 Enter number of messages allowed on link at one time.
1
Building Link number          2
 ENTER LINK NAME
L2
 Enter number of messages allowed on link at one time.
1
** Processing ***.  statement     1 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          1
** Processing ***.  statement     3 PROCESS          1
** Processing ***.  statement     4 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     6 PROCESS          1
 No such Label on Label List
** Processing ***.  statement     7 PROCESS          1
** Processing ***.  statement     8 PROCESS          1
** Processing ***.  statement     9 PROCESS          1
** Processing ***.  statement    10 PROCESS          1
ERROR *** COULD NOT FIND MSG M2
** Processing ***.  statement    11 PROCESS          1
** Processing ***.  statement    12 PROCESS          1
** Processing ***.  statement    13 PROCESS          1
 No such Label on Label List
** Processing ***.  statement    14 PROCESS          1
 No such Label on Label List
** Processing ***.  statement    15 PROCESS          1
** Processing ***.  statement    16 PROCESS          1
** Processing ***.  statement    17 PROCESS          1
** Processing ***.  statement    18 PROCESS          1
** Processing ***.  statement    19 PROCESS          1
** Processing ***.  statement     1 PROCESS          2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS          2
 No such Label on Label List
** Processing ***.  statement     3 PROCESS          2
** Processing ***.  statement     4 PROCESS          2
** Processing ***.  statement     5 PROCESS          2
 No such Label on Label List
** Processing ***.  statement     6 PROCESS          2
 ERROR---Could not find msg type **  M2        ** in UNLESS
```

```
** Processing ***.  statement     7 PROCESS           2
** Processing ***.  statement     8 PROCESS           2
** Processing ***.  statement     9 PROCESS           2
** Processing ***.  statement    10 PROCESS           2
** Processing ***.  statement    11 PROCESS           2
 ERROR---Could not find msg type **  M2          ** in UNLESS
 No such Label on Label List
** Processing ***.  statement    12 PROCESS           2
** Processing ***.  statement    13 PROCESS           2
** Processing ***.  statement    14 PROCESS           2
** Processing ***.  statement    15 PROCESS           2
** Processing ***.  statement    16 PROCESS           2
** Processing ***.  statement    17 PROCESS           2
** Processing ***.  statement    18 PROCESS           2
** Processing ***.  statement    19 PROCESS           2
  E-NET built
  *** EXERCISENET PROCESSING ***
 ** EXERCISE NET MENU **

  1 - AUTO EXECUTION
  2 - MANUAL EXECUTION
  3 - EXIT
 Enter Option Number.
3
      ***  MASTER MENU  ***


  1 - ENTER MSG FORMAT MENU.
  2 - ENTER PPML FORMAT MENU.
  3 - ENTER EVALUATE PROTOCOL MENU.


 Enter the number corresponding to the desired function.
1
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
4
 MESSAGE LIST
ACK
FIELD NAME  MIN  MAX  CONTENTS
ERROR         0    0         0
F1           20   20         1
MESSAGE TYPE
M1
FIELD NAME  MIN  MAX  CONTENTS
ERROR         0    0         0
```

```
F1          1000 2000          1
MESSAGE TYPE
M1
FIELD NAME  MIN  MAX  CONTENTS
ERROR         0    0         0
F1          1000 2000          1
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
1
  *** NEW MSG PROCESSING ***

 1- Enter the message type(10 or less characters)
M2
 ** Field Processing for Message Type-- M2
 Enter the Name of the Field.
F1
 Enter the minimun,maximum size of the field in bytes.
1000
1000
 More fields to input?(y or n)
n
     *** MASTER MESSAGE FORMAT MENU ***


  1 - Enter New Message Type.
  2 - Change a Message Format.
  3 - Delete a Message.
  4 - List Message Types.
  5 - Go to Master Menu.

 Enter the number corresponding to the desired function.
5
 Going to Master menu.
      ***  MASTER MENU  ***


  1 - ENTER MSG FORMAT MENU.
  2 - ENTER PPML FORMAT MENU.
  3 - ENTER EVALUATE PROTOCOL MENU.


 Enter the number corresponding to the desired function.
3
  ** LINK PROCESSING **
Building Link number          1
 ENTER LINK NAME
```

```
L1
 Enter number of messages allowed on link at one time.
1
Building Link number        2
 ENTER LINK NAME
L2
 Enter number of messages allowed on link at one time.
2
** Processing ***.  statement     1 PROCESS           1
 No such Label on Label List
** Processing ***.  statement     2 PROCESS           1
** Processing ***.  statement     3 PROCESS           1
** Processing ***.  statement     4 PROCESS           1
 No such Label on Label List
** Processing ***.  statement     5 PROCESS           1
 No such Label on Label List
** Processing ***.  statement     6 PROCESS           1
 No such Label on Label List
** Processing ***.  statement     7 PROCESS           1
** Processing ***.  statement     8 PROCESS           1
** Processing ***.  statement     9 PROCESS           1
** Processing ***.  statement    10 PROCESS           1
** Processing ***.  statement    11 PROCESS           1
** Processing ***.  statement    12 PROCESS           1
** Processing ***.  statement    13 PROCESS           1
 No such Label on Label List
** Processing ***.  statement    14 PROCESS           1
 No such Label on Label List
** Processing ***.  statement    15 PROCESS           1
** Processing ***.  statement    16 PROCESS           1
** Processing ***.  statement    17 PROCESS           1
** Processing ***.  statement    18 PROCESS           1
** Processing ***.  statement    19 PROCESS           1
** Processing ***.  statement     1 PROCESS           2
 No such Label on Label List
** Processing ***.  statement     2 PROCESS           2
 No such Label on Label List
** Processing ***.  statement     3 PROCESS           2
** Processing ***.  statement     4 PROCESS           2
** Processing ***.  statement     5 PROCESS           2
 No such Label on Label List
** Processing ***.  statement     6 PROCESS           2
** Processing ***.  statement     7 PROCESS           2
** Processing ***.  statement     8 PROCESS           2
** Processing ***.  statement     9 PROCESS           2
** Processing ***.  statement    10 PROCESS           2
** Processing ***.  statement    11 PROCESS           2
 No such Label on Label List
** Processing ***.  statement    12 PROCESS           2
** Processing ***.  statement    13 PROCESS           2
** Processing ***.  statement    14 PROCESS           2
** Processing ***.  statement    15 PROCESS           2
** Processing ***.  statement    16 PROCESS           2
** Processing ***.  statement    17 PROCESS           2
** Processing ***.  statement    18 PROCESS           2
** Processing ***.  statement    19 PROCESS           2
  E-NET built
  *** EXERCISENET PROCESSING ***
 ** EXERCISE NET MENU **

  1 - AUTO EXECUTION
  2 - MANUAL EXECUTION
  3 - EXIT
 Enter Option Number.
1
 Auto execution processing
 Enter number of events to process.
100
 RCV Processing here **
 Set Processing here **
 Unless Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 Linkend Processing here **
 Do you want msg number ,         1
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Set Processing here **
 Assign Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 If  Processing here **
 Linkend Processing here **
 Do you want msg number ,         2
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Goto Processing here **
 Goto Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Unless Processing here **
 Goto Processing here **
 RCV Processing here **
 Set Processing here **
 Unless Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 Linkend Processing here **
 Do you want msg number ,          3
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
Y
 SERVICE LINK  Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Set Processing here **
 Assign Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 If  Processing here **
 Linkend Processing here **
 Do you want msg number ,          4
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Goto Processing here **
 Goto Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Unless Processing here **
 Goto Processing here **
 RCV Processing here **
 Set Processing here **
 Unless Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 Linkend Processing here **
 Do you want msg number ,          5
 to be lost?(Y/N)
N
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Set Processing here **
 Assign Processing here **
 Send Processing here **
 SERVICE LINK  Processing here **
 If  Processing here **
 Linkend Processing here **
 Do you want msg number ,         6
 to be lost?(Y/N)
Y
 Do you want the message to have an error in it?(Y/N)
N
 SERVICE LINK  Processing here **
 Goto Processing here **
 Goto Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 If  Processing here **
 Unless Processing here **
 Goto Processing here **
 Unless Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Assign Processing here **
 Unless Processing here **
 If  Processing here **
 RCV Processing here **
 Goto Processing here **
 Unless Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 RCV Processing here **
 Assign Processing here **
 Unless Processing here **
 If  Processing here **
 Unless Processing here **
 Goto Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
```

```
   5 - CONTINUE EXECUTION
   6 - SHOW GLOBAL VAR LIST
   7 - SHOW E-NET STRUCTURE
  10 - EXIT
 Enter Option Number
3
 **  ACTMSGLIST  **
 NUMBER |   TYPE     | SIZE | TIME SENT | TIME RCVD   |  FLD NAME   | C
 ----------------------------------------------------
     6  | ACK        |   20 |     28.00 |      0.00
                                                          ERROR
                                                          F1
     5  | M1         | 2000 |     23.00 |     26.00
                                                          ERROR
                                                          F1
     4  | ACK        |   20 |     17.00 |     20.00
                                                          ERROR
                                                          F1
     3  | M2         | 1000 |     12.00 |     15.00
                                                          ERROR
                                                          F1
     2  | ACK        |   20 |      6.00 |      9.00
                                                          ERROR
                                                          F1
     1  | M1         | 2000 |      1.00 |      4.00
                                                          ERROR
                                                          F1
     0  | TOKEN      |    1 |      0.00 |      0.00
 ** MANUAL NET EXECUTION MENU **

   1 - SHOW TOKEN MACHINE
   2 - SHOW EVENTLIS
   3 - SHOW ACTIVE MESSAGE LIST
   4 - SHOW TOKEN PLACEMENT
   5 - CONTINUE EXECUTION
   6 - SHOW GLOBAL VAR LIST
   7 - SHOW E-NET STRUCTURE
  10 - EXIT
 Enter Option Number
6
 *** GLOBAL VARIABLE LISTING ***

   Var Name          Value
WAITCTR                    16
CLOCK                     100
 ** MANUAL NET EXECUTION MENU **

   1 - SHOW TOKEN MACHINE
   2 - SHOW EVENTLIS
   3 - SHOW ACTIVE MESSAGE LIST
   4 - SHOW TOKEN PLACEMENT
   5 - CONTINUE EXECUTION
   6 - SHOW GLOBAL VAR LIST
   7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
2
 ** SHOW EVENTLIST PROCESSING HERE **
 TIME | TRANSNUM | PROC | STMT
______________________________
 48.00          47       2      15
 48.00          13       1       7
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
10
Exiting
 ** EXERCISE NET MENU **

  1 - AUTO EXECUTION
  2 - MANUAL EXECUTION
  3 - EXIT
 Enter Option Number.
1
 Auto execution processing
 Enter number of events to process.
3
 Unless Processing here **
 Assign Processing here **
 RCV Processing here **
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
2
 ** SHOW EVENTLIST PROCESSING HERE **
 TIME | TRANSNUM | PROC | STMT
______________________________
 49.00          14       1       8
 50.00          42       2      11
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
5
 If  Processing here **
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
10
Exiting
 ** EXERCISE NET MENU **

  1 - AUTO EXECUTION
  2 - MANUAL EXECUTION
  3 - EXIT
 Enter Option Number.
20
 ** EXERCISE NET MENU **

  1 - AUTO EXECUTION
  2 - MANUAL EXECUTION
  3 - EXIT
 Enter Option Number.
1
 Auto execution processing
 Enter number of events to process.
20
 Unless Processing here **
 Goto Processing here **
 Unless Processing here **
 RCV Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 Assign Processing here **
 Unless Processing here **
 If  Processing here **
 RCV Processing here **
 Goto Processing here **
 Unless Processing here **
 RCV Processing here **
 Unless Processing here **
 Unless Processing here **
 RCV Processing here **
 Assign Processing here **
 Unless Processing here **
 If  Processing here **
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
6
 *** GLOBAL VARIABLE LISTING ***

  Var Name           Value
WAITCTR                   13
CLOCK                    100
 ** MANUAL NET EXECUTION MENU **

  1 - SHOW TOKEN MACHINE
  2 - SHOW EVENTLIS
  3 - SHOW ACTIVE MESSAGE LIST
  4 - SHOW TOKEN PLACEMENT
  5 - CONTINUE EXECUTION
  6 - SHOW GLOBAL VAR LIST
  7 - SHOW E-NET STRUCTURE
 10 - EXIT
 Enter Option Number
>
```

Vita

Captain Kenneth R. Martin Jr. was born September 12, 1949. After graduating from San Diego State University with a B.A. degree in Mathematics, he joined the Air Force and was sent to Officer Training School. He was commissioned in August 1972. He was then sent to Navigator Training School. After graduating from Navigator Training School, he was sent to Macdill AFB for training as a Weapons System Officer in the F-4. He flew the F-4 at Udorn RTAFB and RAF Woodbridge. He was then reassigned to Mountain Home AFB wher he flew the F-111A for four years. He received an assignment to AFIT in June 1982. Captain Martin is unmarried with two children, Julie Rene and David Michael.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/EE/83D-12	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL *(If applicable)* AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State and ZIP Code)* Air Force Institute of Technology Wright Patterson AFB, Oh 45433	7b. ADDRESS *(City, State and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State and ZIP Code)*	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE *(Include Security Classification)* See box 19	

12. PERSONAL AUTHOR(S) Martin, Kenneth R., Jr., B.A., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
M.S. Thesis	FROM ______ TO ______	1983, Dec	216

16. SUPPLEMENTARY NOTATION

Approved for public release: IAW AFR 190-17.
[illegible] 7 Feb 84
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	Computer Networks, Protocol Verification, Evaluation Net, Program Process Modeling Language
09	02		

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Title: An Automated Computer Communication Network Protocol Verification System

Thesis Advisor: Maj. Walter D. Seward, USAF, EE Dept

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER *(Include Area Code)*	22c. OFFICE SYMBOL
Maj. Walter D. Seward	(513)-255-3450	AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

Abstract

An automated tool for computer network communication protocol validation was developed and implemented. The method utilyzes the Program Process Modeling Language to specify the protocol and an automated procedure to convert the description into an equivalent Evaluation Net(E-Net) in order to evaluate the protocol. Simulation techniques are used to exercise the E-Net presenting data on message transmission times and global state generation.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAG

FILMED